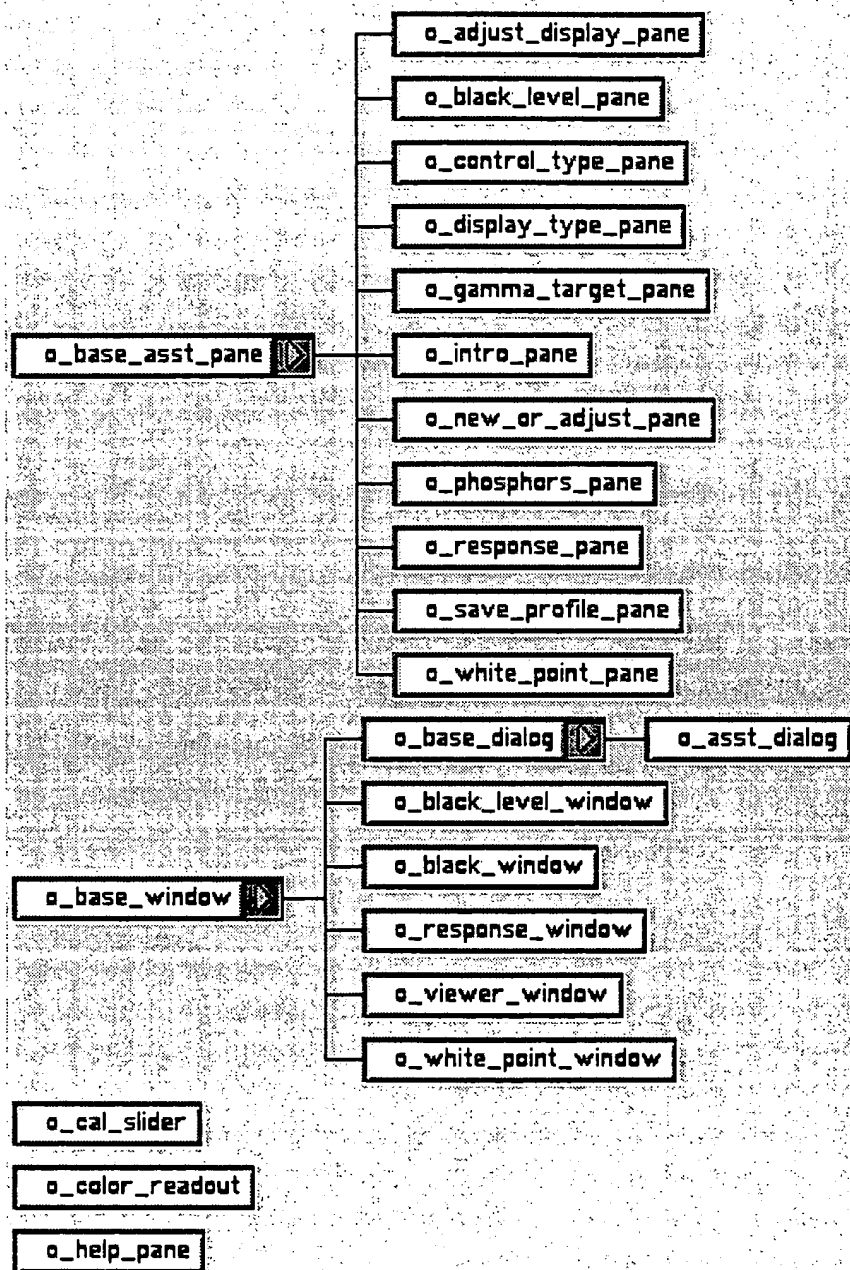


EXhibit B

345 Pages follow

File	Code	Data		
Main	9K	1K	•	•
main.cp	9K	1K	•	•
globals.h	0	0	•	•
directives_carbon.h	0	0	•	•
directives_os8.h	n/a	n/a	•	•
Robert's	35K	2K	•	•
gamma_utils.cp	3K	288	•	•
cal_math.cp	26K	1K	•	•
illus_out.c	5K	919	•	•
Windows	28K	4K	•	•
o_asst_dialog.cp	6K	1K	•	•
o_black_window.cp	1K	391	•	•
o_black_level_window.cp	3K	707	•	•
o_response_window.cp	7K	902	•	•
o_viewer_window.cp	2K	441	•	•
o_white_point_window.cp	4K	682	•	•
o_color_readout.cp	4K	387	•	•
Panes	34K	5K	•	•
o_base_asst_pane.cp	884	277	•	•
o_intro_pane.cp	624	260	•	•
o_new_or_adjust_pane.cp	5K	816	•	•
o_display_type_pane.cp	1020	267	•	•
o_control_type_pane.cp	1K	267	•	•
o_adjust_display_pane.cp	1K	571	•	•
o_black_level_pane.cp	1K	488	•	•
o_response_pane.cp	1K	352	•	•
o_white_point_pane.cp	1K	359	•	•
o_gamma_target_pane.cp	1K	431	•	•
o_phosphors_pane.cp	1K	326	•	•
o_save_profile_pane.cp	8K	1K	•	•
o_cal_slider.cp	6K	320	•	•
o_help_pane.cp	2K	233	•	•
Common Code	64K	14K	•	•
o_base_dialog.cp	2K	618	•	•
o_base_window.cp	7K	978	•	•
my_alerts.c	1K	138	•	•
my_apple_events.c	1K	88	•	•
my_colorsync.c	2K	3K	•	•
my_controls.c	352	67	•	•
my_dialogs.c	10K	1K	•	•
my_displays.c	4K	304	•	•
my_files.c	7K	457	•	•
my_gestalts.c	5K	2K	•	•
my_macros.h	0	0	•	•
my_menus.c	516	87	•	•
my_quickdraw.c	10K	1K	•	•
my_strings.c	3K	205	•	•
my_utilities.c	4K	2K	•	•
my_windows.c	1K	212	•	•
Resources	0	0	•	•
main.rsrc	n/a	n/a	•	•
carb.rsrc	n/a	n/a	•	•
plst.rsrc	n/a	n/a	•	•
InfoPlist.plc	n/a	n/a	•	•
129.icns	n/a	n/a	•	•
help_copy.sh	n/a	n/a	•	•
Libraries	16K	3K	•	•
console.stubs.c	340	80	•	•
MSL_Runtime_PPC.Lib	16K	3K	•	•
Libraries - Carbon	103K	23K	•	•
MSL_C_Carbon.Lib	103K	23K	•	•
CarbonLib	0	0	•	•
Libraries - OS8	0	0	•	•
CarbonAccessors.o	n/a	n/a	•	•
MSL_C_PPC.Lib	n/a	n/a	•	•

File	Code	Data
InterfaceLib	n/a	n/a
AppearanceLib	n/a	n/a
ColorSyncLib	n/a	n/a
ControlsLib	n/a	n/a
DisplayLib	n/a	n/a
MathLib	n/a	n/a
MenusLib	n/a	n/a
QuickTimeLib	n/a	n/a
WindowsLib	n/a	n/a




```
//  
// _____ ©1998-2001 bergdesign inc.  
//  
  
#ifdef __APPLE_CC_  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Displays.h>  
#include <Sound.h>  
#include <Types.h>  
#include <Memory.h>  
#include <Quickdraw.h>  
#include <Fonts.h>  
#include <Events.h>  
#include <Menus.h>  
#include <Windows.h>  
#include <TextEdit.h>  
#include <Dialogs.h>  
#include <OSUtils.h>  
#include <ToolUtils.h>  
#include <SegLoad.h>  
#include <Sound.h>  
#endif  
#endif  
  
#include "globals.h"  
  
#include "my_apple_events.h"  
#include "my_gestalts.h"  
#include "my_menus.h"  
#include "my_display.h"  
  
#include "o_asst_dialog.h"  
#include "o_response_window.h"  
#include "o_black_window.h"  
  
#include "cal_math.h"  
#include "gamma_utils.h"  
  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
// Required functions  
pascal Boolean CanCalibrate ( AVIDType );  
pascal OSErr Calibrate ( CalibratorInfo * );  
  
#ifdef __cplusplus  
}  
#endif  
  
// Our functions  
  
void do_Gestalt_Checks ( void );  
int do_Init_Variables ( struct cal_globals * );  
void do_Kill_Variables ( struct cal_globals * );  
  
OSStatus do_Fetch_Tri_Values ( struct cal_globals * );  
void do_Debug_Tri_Values ( struct cal_globals * );  
void do_Dump_Tri_Values ( struct cal_globals * );  
  
void do_Install_HI_Command_Handler( void );  
void do_Remove_HI_Command_Handler( void );  
pascal OSStatus do_Process_HI_Command ( EventHandlerCallRef, EventRef, void* );  
  
void do_Main_Event_Loop ( struct cal_globals * );  
void do_Handle_Event ( EventRecord *, struct cal_globals * );  
void do_Handle_Mouse_Event ( EventRecord *, struct cal_globals * );  
void do_Menu_Command( long , struct cal_globals * );  
void do_Handle_Key_Event ( EventRecord *, struct cal_globals * );  
void do_Handle_Update_Event ( EventRecord *, struct cal_globals * );  
void do_Handle_Activate_Event ( EventRecord *, struct cal_globals * );  
void do_Handle_OS_Event ( EventRecord *, struct cal_globals * );  
void do_Handle_High_Level_Event ( EventRecord *, struct cal_globals * );  
  
void do_Quit_App( struct cal_globals * );  
  
Boolean do_Get_Object_Ptr_From_Window ( WindowRef, o_base_window ** );  
  
OSStatus do_Install_AE_Handlers( struct cal_globals * );
```

```
OSStatus      do_Remove_AE_Handlers( void );

OSStatus      do_AE_DM_Notification( AppleEvent *, AppleEvent *, long );
OSErr         do_Handle_DM_Notification( AppleEvent * );
OSStatus      do_AE_Show_Preferences( AppleEvent *, AppleEvent *, long );
OSStatus      do_AE_Quit_App( AppleEvent *, AppleEvent *, long );
OSStatus      do_AE_Not_Handled( AppleEvent *, AppleEvent *, long );

OSErr         do_Build_Profile_List ( struct cal_globals * );
void          do_Dispose_Profile_List ( struct cal_globals * );
static pascal OSErr do_Profile_Iterate ( CMPProfileIterateData *, void * );
```

```

// _____ ©1998-2001 bergdesign inc.
//
#include "main.h"

#ifdef __APPLE_CC__
#include <ApplicationServices/ApplicationServices.h>
#else
#include "CMDeviceIntegration.h"
#endif

DECLARE_DEBUG_FILE_PTR;

EventHandlerUPP          gProcessHICommandUPP = NULL;
CMPProfileIterateUPP     profile_iterate_proc = NULL;

// _____

int main(void)
{
    OSStatus              err = noErr;

    OPEN_DEBUG_FILE("supercal_log", "w");

    do_Init_Toolbox();
    do_Gestalt_Checks ();
    // do_Install_AE_Handlers();
    // do_Init_Variables ();
    do_Init_MenuBar ();
    do_Init_Help();

    // Init some of the CalibratorInfo usually done by the
    // Monitors control panel that calls the shared library
    CalibratorInfo theInfo;
    CMPProfileLocation prof_loc;
    theInfo.profileLocationSize = 0;
    theInfo.profileLocationPtr = &prof_loc;
    theInfo.eventProc = NULL;
    theInfo.isGood = false;

    // We start by opening our app window onto the main display and getting that one.
    // Once it's open, we'll let the user drag it onto another display.
    GDHandle main_device = GetMainDevice();
    DMGetDisplayIDByGDevice( main_device, &(theInfo.displayID), false );
    DEBUG_VAR_PRINT("Main Device DisplayID: %d", theInfo.displayID);
    DEBUG_EXTRA_VAR_PRINT(" GDevice: 0x%X", *main_device);

    // OSStatus GetWindowGreatestAreaDevice( WindowRef inWindow, WindowRegionCode inRegion, GDHandle
    *outGreatestDevice, Rect *outGreatestDeviceRect);

    // CMDeviceProfileID  the_profile;
    // CMGetDeviceDefaultProfileID( cmDisplayDeviceClass, theInfo.displayID, &the_profile );

    // do_Install_HI_Command_Handler();

    Calibrate( &theInfo );

    // If the calibration was successful
    if( theInfo.isGood && ( NULL != theInfo.profileLocationPtr ) )
    {
        CMPProfileRef    prof_ref = NULL;

        err = CMOpenProfile( &prof_ref, theInfo.profileLocationPtr );
        if( prof_ref != NULL )
        {
            err = CMSetProfileByAVID( theInfo.displayID, prof_ref );
            CMCloseProfile( prof_ref );
        }
    }

    // do_Remove_HI_Command_Handler();

    CLOSE_DEBUG_FILE;

    return( err );
}

// _____

pascal Boolean CanCalibrate ( AVIDType display_id )
{
#pragma unused ( display_id )

    // Here you should determine if your calibrator can work
    // on the display indicated by the AVIDType passed in.

```

```
// There are a host of Display Manager calls that allow
// you to get all sorts of information about a monitor
// requiring only the AVIDType.

// If your calibrator requires additional hardware or additional
// code libraries - you ought to check for these at this time
// and return false if you cannot operate.

// Do not put up any alerts or dialogs here! No UI!
// If you must present a message to the user it's better that
// you "lie" here and return true - you can then put up your
// alert if the user selects your calibrator from the list.

return( true );
}

//
//
pascal OSErr Calibrate ( CalibratorInfo *theInfo )
{
    OSErr err = noErr;
    struct cal_globals globals;
    struct graphics_dev_info main_dev_info, saved_dev_info;
    struct test_point_info *test_points;
    struct control_point_info control_points_r;
    struct control_point_info control_points_g;
    struct control_point_info control_points_b;
    struct scale_info plot_scale;
    struct scale_info wp_scale;

    // Grab the display ID and update proc from the info passed in by
    // the Monitors control panel, or our own main() routine.
    globals.display_id = theInfo->displayID;
    globals.event_proc = theInfo->eventProc;

    // Get memory for the control structures full of arrays for each color component
    err = get_control_point_memory(&control_points_r);
    if ( err )
        goto bail;

    err = get_control_point_memory(&control_points_g);
    if ( err )
        goto bail;

    err = get_control_point_memory(&control_points_b);
    if ( err )
        goto bail;

    // Get memory for the array of test point structures (only need one set because we calibrate one component at a
    // time)
    err = ( 0 == (test_points = (struct test_point_info *)calloc( 1, sizeof(struct test_point_info) *
    MAX_TEST_POINTS )) );
    if ( err )
        goto bail; // Not enough ram for test points

    // Initialize some things in the component structure
    globals.this_component.this_dev_info = &main_dev_info;
    globals.this_component.saved_dev_info = &saved_dev_info;

    err = do_Init_Variables ( &globals );
    if ( err )
        goto bail;

    do_Build_Profile_List( &globals );

    do_Fetch_Tri_Values( &globals );
    do_Debug_Tri_Values( &globals );

    globals.this_component.plot_scale = &plot_scale;
    globals.this_component.wp_scale = &wp_scale;

    globals.this_component.test_points = test_points;

    globals.this_component.max_pixel_value = 255.0;

    globals.this_component.gam_tab_count = globals.this_component.this_dev_info->entry_count;
    globals.this_component.gam_tab_max_value = globals.this_component.this_dev_info->max_value;

    globals.this_component.cp_r = &control_points_r;
    globals.this_component.cp_g = &control_points_g;
    globals.this_component.cp_b = &control_points_b;

    // Bring up the main assistant window and init associated variables
    globals.asst_dialog = new o_asst_dialog ( kAssistantDialogDLOG, &globals );
    if ( globals.asst_dialog == NULL )
        goto bail;

    do_Install_AE_Handlers( &globals );
}
```

```
// Fall into event loop until user chooses to leave.
do_Main_Event_Loop( &globals );
// RunApplicationEventLoop();

do_Remove_AE_Handlers();

// The two fields of the CalibratorInfo struct that a
// calibrator must fill in are the 'isGood' field and
// the 'profileLocation'. If isGood == true, the profile
// returned will be made the default profile for the current
// monitor. The profileLocation simply points to the
// profile.
// It is advised that you ensure the profileLocation is:
//   a: a file-based profile (not handle based or other)
//   b: located somewhere within the ColorSync Profiles folder.
// Otherwise, the newly created profile may fail to show
// up in the Monitors and Sound ColorSync Profile list.

// Flag the profile as good (the user clicked "Okay").
theInfo->isGood = globals.create_profile;

if ( theInfo->isGood && theInfo->profileLocationPtr != NULL )
{
    *(theInfo->profileLocationPtr) = globals.chosen_profile_loc;
}
else
{
    copy_gamma_to_dev( globals.this_component.saved_dev_info );
}

bail:

if ( globals.asst_dialog )
{
    delete (globals.asst_dialog);
    globals.asst_dialog = NULL;
}

if ( globals.response_window )
{
    delete (globals.response_window);
    globals.response_window = NULL;
}

if ( globals.black_level_window )
{
    delete (globals.black_level_window);
    globals.black_level_window = NULL;
}

if ( globals.white_point_window )
{
    delete (globals.white_point_window);
    globals.white_point_window = NULL;
}

do_Dispose_Profile_List( &globals );

do_Dump_Tri_Values( &globals );

do_Kill_Variables ( &globals );

if ( test_points )
    free(test_points);

dump_control_point_memory(&control_points_b);
dump_control_point_memory(&control_points_g);
dump_control_point_memory(&control_points_r);

return ( err );
}

//
#pragma mark -
//

void do_Gestalt_Checks ( void )
{
    OSErr      err = noErr;

#if TARGET_API_MAC_OS8

    // If we're under Classic, we don't have access to hardware like
    // we do under Mac OS 8 & 9.
    if ( do_Check_If_Running_On_Classic() )
```

```
{
    do_One_Button_Alert( kAlertNoteAlert, "\pThis version of SuperCal will not run under Classic.",
        "\pPlease download the Carbon version for native use under Mac OS X or under Classic.", "\pOK"
    );
    ExitToShell();
}

if ( !do_Check_For_System_Version ( 0x0860 ) )
    ExitToShell();

// We check for ColorSync 2.6
if ( !do_Check_For_ColorSync ( 0x0260 ) )
    ExitToShell();

#else if TARGET_API_MAC_CARBON

    // If we use any Mac OS X functionality not present in OS 9 or earlier,
    // we would use this to check for OS X.
    // do_Check_If_Running_On_Carbon_X();

    if ( !do_Check_For_System_Version ( 0x0900 ) )
    // if ( !do_Check_For_System_Version ( 0x1152 ) ) // test code
        ExitToShell();

    if ( !do_Check_For_Carbon_Version( 0x0140 ) )
    // if ( !do_Check_For_Carbon_Version( 0x0383 ) ) // test code
        ExitToShell();

    if ( !do_Check_For_ColorSync ( 0x0300 ) )
    // if ( !do_Check_For_ColorSync ( 0x0516 ) ) // test code
        ExitToShell();

#endif

    // DMGetGDeviceByDisplayID() needs InterfaceLib 7.5 or later
    if ( !do_Check_For_Display_Manager ( 0x0225 ) )
        ExitToShell();

    // We register ourselves as an appearance client in the gestalt check.
    if ( !do_Check_For_Appearance_Manager ( 0x0101 ) )
    // if ( !do_Check_For_Appearance_Manager ( 0x0287 ) ) // test code
        ExitToShell();

    if ( !do_Check_For_Drag_Manager ( 0x0100 ) )
        ExitToShell();

    if ( !do_Check_For_Apple_Events ( 0x0100 ) )
        ExitToShell();

    Boolean mirroring_on = false;
    err = DMIsMirroringOn( &mirroring_on );
    if( mirroring_on )
    {
        do_One_Button_Alert( kAlertNoteAlert, "\pSuperCal will not function properly when video mirroring is turned
on",
            "\pPlease turn video mirroring off and make the mirrored display your main display if you wish
calibrate it.", "\pOK" );
    }
}

//
int do_Init_Variables ( struct cal_globals *globals )
{
    OSErr err = noErr;

    // Since we're running in someone else's heap space, we MUST
    // initialize all of our variables. We cannot rely on the compiler
    // to insure that our variables are initialized to NULL.

    globals->user_done = false;

    globals->asst_dialog = NULL;
    globals->create_profile = false;

    globals->expert_mode = true;
    globals->number_of_channels = 3;

    // new or adjust
    globals->profile_loc_array = NULL;
    globals->chosen_profile_loc.locType = cmNoProfileBase;
    globals->chosen_profile_index = -1;
    globals->number_of_profiles = 0;
    globals->count = 0;
    globals->profile_menu = NULL;

    // debug statements
    DisplayVideoSettings();
    GDHandle device_handle = NULL;
```

```
err = DMGetGDeviceByDisplayID( globals->display_id, &device_handle, false );
struct display_specs the_specs;
do_Get_Display_Specs( globals->display_id, &the_specs );

// do_Test_Display_APIS();

// display type
if( 0 == the_specs.frequency )
    globals->display_type = kDisplayTypeLCD;
else
    globals->display_type = kDisplayTypeCRT;

// type of controls
globals->controls_type = 0;

// adjust display
globals->black_window = NULL;

// black level
globals->black_level_window = NULL;
globals->black_level_complete = false;

// response
globals->response_window = NULL;
globals->response_complete = false;
globals->viewer_window = NULL;

// white point
globals->white_point_window = NULL;
globals->white_point_complete = false;

// chromaticities
globals->tri_choice = 0;
globals->rez_data = NULL;
globals->tri_count = 0;
globals->tri_data = NULL;

// target gamma
globals->target_gamma = 1.8;
globals->target_perceptual = false;

// save profile
globals->profile_file_name[0] = 0;
globals->profile_name[0] = 0;

globals->this_component.this_dev_info->display_id = globals->display_id;
globals->this_component.saved_dev_info->display_id = globals->display_id;

GDHandle device = NULL;
err = DMGetGDeviceByDisplayID ( globals->display_id, &device, false );
if ( err )
    goto bail;

globals->display_bounds = do_Get_Display_Bounds_From_GDHandle( device );

err = malloc_graphics_dev_info_vcg(globals->this_component.this_dev_info);
if( err )
    goto bail;

err = malloc_graphics_dev_info_vcg(globals->this_component.saved_dev_info);
if( err )
    goto bail;

err = copy_gamma_from_dev(globals->this_component.saved_dev_info);

err = test_graphics_dev(globals->this_component.this_dev_info);
if( err )
    goto bail;

// err = copy_graphics_dev_info( globals->this_component.saved_dev_info, globals->this_component.this_dev_info );

bail:

    return err;
}

//
void do_Kill_Variables ( struct cal_globals *globals )
{
    if ( globals->this_component.this_dev_info )
        free_graphics_dev_info_vcg(globals->this_component.this_dev_info);

    if ( globals->this_component.saved_dev_info )
        free_graphics_dev_info_vcg(globals->this_component.saved_dev_info);
}

//
```

```
#pragma mark -

//
OSStatus do_Fetch_Tri_Values ( struct cal_globals *globals )
{
    OSStatus          err = noErr;
    char              *rez_ptr = NULL;
    unsigned long      offset = 0;
    int               i = 0;

    globals->rez_data = GetResource( 'tri#', 128 );
    if( globals->rez_data == NULL )
        return( ResError() );
    else
        HLock( globals->rez_data );

    rez_ptr = *(globals->rez_data);

    // first long is the number of tristimulus data sets
    globals->tri_count = *((unsigned long *)rez_ptr);
    rez_ptr += A_LONG;

    // grab some memory to hold the pointer structure
    globals->tri_data = (struct tri_info *)malloc( globals->tri_count * sizeof(struct tri_info) );
    if( globals->tri_data != NULL )
    {
        for( i=0; i<globals->tri_count; i++ )
        {
            // first long is size of data set, not including the size
            unsigned long size = *((unsigned long *)rez_ptr);
            rez_ptr += A_LONG;
            char *next_rez = rez_ptr + size;

            globals->tri_data[i].manufacturer = (unsigned char *)rez_ptr;
            rez_ptr += A_CHAR + *rez_ptr;

            globals->tri_data[i].model = (unsigned char *)rez_ptr;
            rez_ptr += A_CHAR + *rez_ptr;

            globals->tri_data[i].variant = (unsigned char *)rez_ptr;
            rez_ptr += A_CHAR + *rez_ptr;

            // padding after the pstrings so the following values are word aligned
            if( ((unsigned long)rez_ptr % 4) != 0 )
                rez_ptr += A_LONG - ( (unsigned long)rez_ptr % 4 ); // Remember, mod operator returns how many byte
we are past the last boundary, so we need the inverse

            globals->tri_data[i].type = *((long *)rez_ptr);
            rez_ptr += A_LONG;

            globals->tri_data[i].values = (struct tri_chrom_values *)rez_ptr;
            rez_ptr = next_rez;
        }
    }
    else
    {
        err = memFullErr;
    }

    return( err );
}

//

void do_Debug_Tri_Values ( struct cal_globals *globals )
{
    char    temp_str[256];
    int     i = 0;
    float   white_X, white_Y, white_Z, white_x, white_y;
    double  temp;

    DEBUG_VAR_PRINT("Number of tri# data: %d",globals->tri_count);

    for( i=0; i<globals->tri_count; i++ )
    {
        DEBUG_VAR_PRINT("----- #d -----",i);
        do_p2c_strcpy( temp_str, globals->tri_data[i].manufacturer );
        DEBUG_VAR_PRINT("manufacturer: %s",temp_str);
        do_p2c_strcpy( temp_str, globals->tri_data[i].model );
        DEBUG_VAR_PRINT("model: %s",temp_str);
        do_p2c_strcpy( temp_str, globals->tri_data[i].variant );
        DEBUG_VAR_PRINT("variant: %s",temp_str);

        if( globals->tri_data[i].values->native_white == 0 ) // 5000
        {

```



```
        white_X = FixedToFloat(globals->tri_data[i].values->white_5000_X);
        white_Y = FixedToFloat(globals->tri_data[i].values->white_5000_Y);
        white_Z = FixedToFloat(globals->tri_data[i].values->white_5000_Z);
    }
    else if ( globals->tri_data[i].values->native_white == 1 ) // 6500
    {
        white_X = FixedToFloat(globals->tri_data[i].values->white_6500_X);
        white_Y = FixedToFloat(globals->tri_data[i].values->white_6500_Y);
        white_Z = FixedToFloat(globals->tri_data[i].values->white_6500_Z);
    }
    else // 9300
    {
        white_X = FixedToFloat(globals->tri_data[i].values->white_9000_X);
        white_Y = FixedToFloat(globals->tri_data[i].values->white_9000_Y);
        white_Z = FixedToFloat(globals->tri_data[i].values->white_9000_Z);
    }

    white_x = white_X / ( white_X + white_Y + white_Z );
    white_y = white_Y / ( white_X + white_Y + white_Z );

    DEBUG_PRINT("native white temp: ");
    if( noErr == do_calc_cct( white_x, white_y, &temp ) )
    {
        DEBUG_EXTRA_VAR_PRINT("%2.1lfK",temp);
    }
    else
    {
        DEBUG_EXTRA_PRINT("error");
    }

    white_X = FixedToFloat(globals->tri_data[i].values->red_X) +
FixedToFloat(globals->tri_data[i].values->green_X) + FixedToFloat(globals->tri_data[i].values->blue_X);
    white_Y = FixedToFloat(globals->tri_data[i].values->red_Y) +
FixedToFloat(globals->tri_data[i].values->green_Y) + FixedToFloat(globals->tri_data[i].values->blue_Y);
    white_Z = FixedToFloat(globals->tri_data[i].values->red_Z) +
FixedToFloat(globals->tri_data[i].values->green_Z) + FixedToFloat(globals->tri_data[i].values->blue_Z);

    white_x = white_X / ( white_X + white_Y + white_Z );
    white_y = white_Y / ( white_X + white_Y + white_Z );

    DEBUG_PRINT("adapted white temp: ");
    if( noErr == do_calc_cct( white_x, white_y, &temp ) )
    {
        if( temp < 5025 && temp > 4975 )
        {
            DEBUG_EXTRA_VAR_PRINT("%2.1lfK ok",temp);
        }
        else
        {
            DEBUG_EXTRA_VAR_PRINT("%2.1lfK is out of range",temp);
        }
    }
    else
    {
        DEBUG_EXTRA_PRINT("error");
    }

    fflush(stderr);
}

//
void do_Dump_Tri_Values ( struct cal_globals *globals )
{
    if( globals->tri_data != NULL )
    {
        free( globals->tri_data );
    }

    if( globals->rez_data != NULL )
    {
        HUnlock( globals->rez_data );
        ReleaseResource( globals->rez_data );
    }
}

//
#pragma mark -
//
OSErr do_Build_Profile_List ( struct cal_globals *globals )
{
    OSErr err = noErr;

    if( globals->profile_menu == NULL )
```

```
{
    globals->profile_menu = GetMenu( kProfileMenuID );
    if( globals->profile_menu != NULL )
    {
        InsertMenu(globals->profile_menu,-1);

        profile_iterate_proc = NewCMPProfileIterateUPP ( do_Profile_Iterate );

        globals->profile_loc_array = NULL;
        globals->chosen_profile_index = -1;
        globals->number_of_profiles = 0;
        globals->count = 0;

        // The first time we iterate thru the profiles, we count how many there are so that
        // we can allocate the right amount of space to hold all of their locations.
        err = CMIterateColorSyncFolder ( profile_iterate_proc, 0L, 0L, globals );
        if ( noErr == err )
        {
            // Here we allocate the space for the profile locations.
            // Checking for memory allocation errors is particularly important
            // since we're just a plug-in piece living in the Monitor's and
            // Sound heap space. If we need more than 500K or so, we ought
            // to use temp mem.

            // Once we know how many profiles there are, we can allocate memory for storing their locations.
            globals->profile_loc_array = (CMPProfileLocation *)NewPtr(sizeof(CMPProfileLocation) *
globals->number_of_profiles);
            if ( NULL != globals->profile_loc_array )
            {
                // This pass stores the locations in the allocated memory.
                err = CMIterateColorSyncFolder ( profile_iterate_proc, 0L, 0L, globals );
                if ( noErr == err )
                {
                    // do_Draw_One_Control_As_DItem( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items );
                }
                else
                {
                    do_Alert_If_Error( "\pError in CMIterateColorSyncFolder() on second pass.", err );
                }
            }
            else
            {
                do_Alert_If_Error( "\pCould not allocate memory for profile data.", MemError() );
            }
        }
        else
        {
            do_Alert_If_Error( "\pError in CMIterateColorSyncFolder() on first pass.", err );
        }
    }
    else
    {
        // menu problem
    }
}
else
{
    return( paramErr );
}

return ( err );
}

//
void do_Dispose_Profile_List ( struct cal_globals *globals )
{
    if( NULL != globals->profile_menu )
    {
        DeleteMenu( kProfileMenuID );
        DisposeMenu( globals->profile_menu );
        globals->profile_menu = NULL;
    }

    if ( NULL != globals->profile_loc_array )
    {
        DisposePtr ( (Ptr)(globals->profile_loc_array) );
        globals->profile_loc_array = NULL;
        globals->chosen_profile_index = -1;
        globals->number_of_profiles = 0;
        globals->count = 0;
    }

    if ( NULL != profile_iterate_proc )
    {
        DisposeCMPProfileIterateUPP( profile_iterate_proc );
        profile_iterate_proc = NULL;
    }
}
```

```
}

//
// This is the function called iteratively by ColorSync. Each time it
// is called, it is passed a profile found in the user's ColorSync
// profiles folder. We want only RGB profiles intended for a display
// (monitor). Since the profile's header is one of the fields passed
// to us in the CMPProfileIterateData structs, we can check the header's
// 'dataColorSpace' and 'profileClass' fields.

static pascal OSErr do_Profile_Iterate ( CMPProfileIterateData *profile_data, void *ref_con )
{
    OSErr          err = noErr;
    struct cal_globals *globals;
    CMPProfileRef   prof_ref = NULL;
    Boolean         found_it = false;

    // Since we have no real global variables, we had to pass in a pointer
    // to the globals set up in the entry routine.
    globals = (struct cal_globals *)ref_con;

    // Is it an RGB profile? Is it a monitor profile? Is it ours?
    if ( ( profile_data->header.dataColorSpace == cmRGBData ) &&
        ( profile_data->header.profileClass == cmDisplayClass ) &&
        ( profile_data->header.creator == 'berg' ) )
    {
        // Now we look to see if it has our resolution independent measurement data
        CMOpenProfile( &prof_ref, &profile_data->location );
        if( prof_ref != NULL )
        {
            err = CMPProfileElementExists ( prof_ref, 'berh', &found_it );
            if( err == noErr && found_it == true )
            {
                // If we don't have an allocated pointer, we know to just count the number of profiles
                // so that we can allocate the right amount of memory to hold them on the next pass.
                if ( NULL == globals->profile_loc_array )
                {
                    globals->number_of_profiles++;
                }
                // If we have an allocated pointer, we begin to stuff the allocated memory with profile
                // locations as though the memory is an array.
                else
                {
                    if ( globals->count < globals->number_of_profiles )
                    {
                        globals->profile_loc_array[globals->count] = profile_data->location;
                        globals->count++;

                        // By putting a little dummy string into the menu first, then changing the text,
                        // we avoid problems with possible meta-characters in the name of what we're
                        // putting in the menu.
                        unsigned long temp_menu_item = 0x03637370; // a 3 char menu item filler in long format
                        AppendMenu ( globals->profile_menu, (StringPtr)&temp_menu_item );
                        short index = CountMenuItems ( globals->profile_menu );
                        SetMenuItemText ( globals->profile_menu, index, profile_data->name );
                    }
                    else
                    {
                        SysBeep(10);
                    }
                }
            }
        }
        CMCloseProfile( prof_ref );
    }

    return ( err );
}

//
// struct CMPProfileLocation
// {
//     short          locType;      ----
//                                     cmNoProfileBase           = 0,
//                                     cmFileBasedProfile          = 1,
//                                     cmHandleBasedProfile         = 2,
//                                     cmPtrBasedProfile            = 3,
//                                     cmProcedureBasedProfile       = 4
//
//     union CMPProfLoc    u;  -----
//                                     struct CMFileLocation        fileLoc;   ----
//                                     struct CMHandleLocation      handleLoc;  ----
//                                     struct CMPtrLocation         ptrLoc;    ----
//                                     struct CMProcedureLocation    procLoc;   ----
//                                     FSSpec                      spec;
//                                     Handle                      h;
//                                     Ptr                        p;
//                                     CMPProfileAccessUPP
//                                     void *
// };
```

```
//  
#pragma mark -  
  
//  
void do_Install_HI_Command_Handler()  
{  
    EventTypeSpec          eventTypes[1];  
  
    eventTypes[0].eventClass = kEventClassCommand;  
    eventTypes[0].eventKind = kEventCommandProcess;  
  
    gProcessHICommandUPP = NewEventHandlerUPP( do_Process_HI_Command );  
    InstallApplicationEventHandler( gProcessHICommandUPP, 1, eventTypes, NULL, NULL );  
}  
  
//  
void do_Remove_HI_Command_Handler()  
{  
    if( NULL != gProcessHICommandUPP )  
    {  
        DisposeEventHandlerUPP( gProcessHICommandUPP );  
        gProcessHICommandUPP = NULL;  
    }  
}  
  
//  
pascal OSStatus do_Process_HI_Command ( EventHandlerCallRef next_handler, EventRef the_event, void* user_data )  
{  
    OSStatus    handled = eventNotHandledErr;  
    HICommand   hi_command;  
  
    GetEventParameter ( the_event, kEventParamDirectObject, typeHICommand, NULL, sizeof(HICommand), NULL,  
    &hi_command );  
  
    switch( hi_command.commandID )  
    {  
        case kHICommandAbout:  
        {  
            // Need a simple alert dialog  
            SysBeep(1);  
            handled = noErr;  
            break;  
        }  
        case kHICommandPreferences:  
        {  
            SysBeep(1);  
            handled = noErr;  
            break;  
        }  
        case kHICommandQuit:  
        {  
            // Remember that a default quit handler is installed for us.  
            // All we need to do is provide an an apple event handler for it.  
            // This will never be called.  
  
            // do_Quit_App();  
            handled = eventNotHandledErr;  
            break;  
        }  
        default:  
        {  
            handled = eventNotHandledErr;  
            break;  
        }  
    }  
  
    // If we don't handle an event here, it will be propogated up to WaitNextEvent().  
    return( handled );  
}  
  
//  
#pragma mark -  
  
//  
void do_Main_Event_Loop ( struct cal_globals *globals )  
{  
    EventRecord          current_event;  
    Boolean              got_event = false;  
    WindowRef           window_ref = NULL;  
    OpaqueWindowRef      *window_obj;
```

```
globals->user_done = false;

while ( !globals->user_done )
{
    got_event = WaitNextEvent ( everyEvent, &current_event, GetCaretTime(), NULL );
    if ( got_event )
    {
        do_Handle_Event ( &current_event, globals );
    }

    window_ref = FrontNonFloatingWindow();
    if ( window_ref && do_Get_Class_From_Window ( window_ref, &window_obj ) )
    {
        window_obj->do_Idle();
        window_obj->do_Update (); // a test
    }
}

// -----
//                                     handle an event

void do_Handle_Event ( EventRecord *event, struct cal_globals *globals )
{
    // DEBUG_PRINT("Called main::do_Handle_Event()");
    // DEBUG_VAR_PRINT("event.what = %#06X",event->what);
    // DEBUG_VAR_PRINT("event.message = %#010X",event->message);
    // DEBUG_VAR_PRINT("event.when = %#010X",event->when);
    // DEBUG_VAR_PRINT("event.where = %d",event->where.h);
    // DEBUG_EXTRA_VAR_PRINT(",%d",event->where.v);
    // DEBUG_VAR_PRINT("event.modifiers = %#06X",event->modifiers);

    switch ( event->what )
    {
        case nullEvent: // 0 - just here for completeness
        {
            break;
        }
        case mouseDown: // 1
        {
            do_Handle_Mouse_Event ( event, globals );
            break;
        }
        case mouseUp: // 2
        {
            break;
        }
        case autoKey: // 5
        case keyDown: // 3
        {
            do_Handle_Key_Event ( event, globals );
            break;
        }
        case keyUp: // 4
        {
            break;
        }
        case updateEvt: // 6
        {
            do_Handle_Update_Event ( event, globals );
            break;
        }
        case diskEvt: // 7
        {
            break;
        }
        case activateEvt: // 8
        {
            do_Handle_Activate_Event ( event, globals );
            break;
        }
        case osEvt: // 15
        {
            do_Handle_OS_Event ( event, globals );
            break;
        }
        case kHighLevelEvent: // 23
        {
            do_Handle_High_Level_Event ( event, globals );
            break;
        }
    }
}

// -----
//                                     mouse event

void do_Handle_Mouse_Event ( EventRecord *event, struct cal_globals *globals )
```

```
{
short      part;
WindowRef  which_window = NULL;
o_base_window *window_obj;

    part = FindWindow ( event->where, &which_window );

    switch ( part )
    {
        case inDesk: // 0
        {
            break;
        }
        case inMenuBar: // 1
        {
            #ifndef CALIBRATOR_SHARED_LIBRARY
                do_Menu_Command ( MenuSelect ( event->where ), globals );
            #endif

            break;
        }
        case inSysWindow: // 2
        {
            // SystemClick ( event, which_window );
            break;
        }
        case inGoAway: // 6
        {
            // We do this here instead of in the class object since the object would have to
            // delete itself, or send a message back to delete itself. This way, we do the last
            // minute check, then delete it if it's ok.
            if ( TrackGoAway ( which_window, event->where ) )
            {
                DEBUG_PRINT("Close button was hit...");

                do_Quit_App( globals );

                /*
                if ( do_Get_Class_From_Window ( which_window, &window_obj ) )
                {
                    globals->user_done = window_obj->do_OK_To_Close();
                }
                */

            }

            break;
        }
        case inContent: // 3
        case inDrag: // 4
        case inGrow: // 5
        case inZoomIn: // 7
        case inZoomOut: // 8
        case inCollapseBox: // 11
        {
            if ( do_Get_Class_From_Window ( which_window, &window_obj ) )
            {
                window_obj->do_Handle_Click ( event, part );
            }

            break;
        }
        default:
        {
            break;
        }
    }
}

// ----- do menu command -----

void do_Menu_Command ( long menuResult, struct cal_globals *globals )
{
    OSStatus      err = noErr;
    short          menu_id;
    short          menu_item;
    MenuRef        menu_ref;
    MenuCommand    command_id;

    STATUS_PRINT("Entered do_Menu_Command()");

    menu_id = HiWord(menuResult);
    menu_item = LoWord(menuResult);
    menu_ref = GetMenuRef(menu_id);
    err = GetMenuItemCommandID( menu_ref, menu_item, &command_id );

    switch( command_id )
    {
        case kHICCommandAbout:
    
```

```
{
    char version_string[32];
    unsigned char app_version[256];

    // Display the app name and version number
    do_Get_App_Vers_Resource_As_CString ( version_string );
    app_version[0] = 0;
    do_p_strcat ( app_version, "\pSuperCal" );
    do_c2p_strcat ( app_version, version_string );

    do_One_Button_Alert( kAlertNoteAlert, app_version, "\pDisplay calibrator for LCDs, CRTs, plasma display
and projectors. Copyright © 1998-2002 bergdesign inc.\nwww.bergdesign.com", "\pOK" );
    break;
}
case kHICCommandPreferences:
{
    SysBeep(1);
    break;
}
case kHICCommandQuit:
{
    do_Quit_App( globals );
    break;
}
default:
{
    break;
}
}

// This turns off highlighting on the menu we just used.
HiliteMenu(0);

STATUS_PRINT("Left do_Menu_Command()");
}

// -----
//                                     key event
void do_Handle_Key_Event ( EventRecord *event, struct cal_globals *globals )
{
    long    the_key;

    the_key = event->message & charCodeMask;
    DEBUG_VAR_PRINT("the_key: %d",the_key);

    if ( ( event->modifiers & cmdKey ) && ( the_key == 'q' || the_key == 'Q' ) )
    {
        do_Quit_App( globals );
    }
    else if ( ( event->modifiers & cmdKey ) && ( the_key == 'w' || the_key == 'W' ) )
    {
        do_Quit_App( globals );
    }
    else
    {
        WindowRef      which_window = FrontNonFloatingWindow();
        o_base_window   *window_obj;

        if ( do_Get_Class_From_Window ( which_window, &window_obj ) )
        {
            window_obj->do_Handle_Key_Down ( event );
        }
    }
}

// -----
//                                     update event
void do_Handle_Update_Event ( EventRecord *event, struct cal_globals *globals )
{
    WindowRef      window_ref;
    o_base_window   *window_obj;

    DEBUG_VAR_PRINT("Got Update event for window %#010X",event->message);

    window_ref = (WindowRef)event->message;

    // Note that we don't have any provisions for windows that are not c++ classes.
    if ( do_Get_Class_From_Window ( window_ref, &window_obj ) )
    {
        DEBUG_PRINT("Event was for our window class");
        window_obj->do_Update ();
    }
    else if ( WindowIsDialog ( window_ref ) )
    {
        DialogRef      dialog;
        SInt16          itemHit;
    }
}
```

```
    DEBUG_PRINT("Event was for a dialog");
    DialogSelect ( event, &dialog, &itemHit );
}
else
{
    #ifdef CALIBRATOR_SHARED_LIBRARY
        DEBUG_PRINT("Event was for our library caller");
        // IMPORTANT!!! Make sure 'eventProc' is not NIL before you call!
        if ( globals->event_proc )
            CallCalibrateEventProc ( globals->event_proc, event );
    #else
        DEBUG_PRINT("Event was not for anything we know about");
        // If we don't know how to handle the update event,
        // we need to at least clear it so it doesn't repeat.
        BeginUpdate ( (WindowPtr)window_ref );
        EndUpdate ( (WindowPtr)window_ref );
    #endif
}
}

// ----- activate event

void do_Handle_Activate_Event ( EventRecord *event, struct cal_globals *globals )
{
    Boolean          activating;
    WindowRef        window_ref;
    o_base_window    *window_obj;
    Boolean          our_window = false;

    // With our implementation of floating windows, we need to work around
    // toolbox routines that generate activate/deactivate events since the
    // window manager assumes that only one window can be active at a time,
    // which is not true with floating windows. SelectWindow, ShowWindow,
    // HideWindow and SendBehind are all functions that implicitly generate
    // activate and deactivate events.
    // With our implementation of floating windows, we install and use an
    // activate handler for each window.

    DEBUG_VAR_PRINT("Got Activate event for window %#010X",event->message);

    window_ref = (WindowRef)event->message;
    activating = ( event->modifiers & activeFlag ) != 0;
    our_window = do_Get_Class_From_Window ( window_ref, &window_obj );

    // Note that we don't have any provisions for windows that are not c++ classes.

    if ( our_window )
    {
        // window_obj->do_Activate_Event_Handler ( window_ref, activating );
        do_Activate_Window ( window_ref, activating );
    }
}
#ifdef CALIBRATOR_SHARED_LIBRARY
else
{
    // IMPORTANT!!! Make sure 'eventProc' is not NIL before you call!
    if ( globals->event_proc )
        CallCalibrateEventProc ( globals->event_proc, event );
}
#endif
}

// ----- os event

void do_Handle_OS_Event ( EventRecord *event, struct cal_globals *globals )
{
    switch ( (event->message >> 24) & 0xFF )
    {
        case mouseMovedMessage:
        {
            break;
        }
        case suspendResumeMessage:
        {
            // Since windows have custom activate handlers, there is no need to
            // call any activate or deactivate routines. The SuspendFloatingWindows()
            // and ResumeFloatingWindows() functions do it properly for us.
            // Remember, we have to avoid any activate events generated by the
            // window manager for our floating windows to function properly.

            if ( event->message & resumeFlag )
            {
                if ( GetMenuBarHeight() > 0 )
                    DrawMenuBar();
                else
                    PaintOne ( NULL, GetGrayRgn() );
            }
        }
    }
}
```



```
        ShowFloatingWindows();
    }
    else
    {
        HideFloatingWindows();
    }
    break;
}
}

// _____ high level event

void do_Handle_High_Level_Event ( EventRecord *event, struct cal_globals *globals )
{
    OSErr    AEResult;

    switch ( event->message )
    {
        case kCoreEventClass:
        {
            AEResult = AEResultAppleEvent(event);    // Handle core AppleEvents
            break;
        }
        default:
        {
            AEResult = AEResultAppleEvent(event);    // Other high level events
            break;
        }
    }
}

// _____

void do_Quit_App( struct cal_globals *globals )
{
    // The black level has to be set before the response can be measured.
    // Likewise, the response has to be measured before the white point can be set.
    if( ( globals->create_profile != true ) && globals->black_level_complete )
    {
        SInt16    answer = kAlertStdAlertCancelButton;

        // We ask the user if they wish to quit.
        answer = do_Two_Button_Alert ( kAlertStopAlert,
                                       "\pAre you sure you want to quit SuperCal?",
                                       "\pYou will lose any measurements that you have made.",
                                       "\pQuit",
                                       "\pContinue" );

        if( kAlertStdAlertOKButton == answer )
        {
            globals->user_done = true;
            QuitApplicationEventLoop();
        }
        else
        {
            globals->user_done = false;
        }
    }
    else
    {
        globals->user_done = true;
    }
}

// _____

#pragma mark -

// _____

OSStatus do_Install_AE_Handlers( struct cal_globals *globals )
{
    OSStatus    err = noErr;

    // We won't bother keeping the routine descriptor pointers around.
    // We'll just retrieve them from the dispatch table when it's time to get rid of them.

    // err += AEInstallEventHandler( kCoreEventClass, kAEOpenApplication, NewAEEEventHandlerUPP(
    (AEEEventHandlerProcPtr)do_AE_Open_App ), 0L, false );
    // err += AEInstallEventHandler( kCoreEventClass, kAEReopenApplication, NewAEEEventHandlerUPP(
    (AEEEventHandlerProcPtr)do_AE_Reopen_App ), 0L, false );
    // err += AEInstallEventHandler( kCoreEventClass, kAEOpenDocuments, NewAEEEventHandlerUPP(
    (AEEEventHandlerProcPtr)do_AE_Open_Doc ), 0L, false );
    // err += AEInstallEventHandler( kCoreEventClass, kAEPrintDocuments, NewAEEEventHandlerUPP(
    (AEEEventHandlerProcPtr)do_AE_Print_Doc ), 0L, false );
}
```

```
// err += AEInstallEventHandler( kCoreEventClass, kAEShowPreferences, NewAEEEventHandlerUPP(
(AEEEventHandlerProcPtr)do_AE_Show_Preferences ), (long)globals, false );
err += AEInstallEventHandler( kCoreEventClass, kAEQuitApplication, NewAEEEventHandlerUPP(
(AEEEventHandlerProcPtr)do_AE_Quit_App ), (long)globals, false );

err += AEInstallEventHandler( typeWildcard, typeWildcard, NewAEEEventHandlerUPP(
(AEEEventHandlerProcPtr)do_AE_Not_Handled ), (long)globals, false );

    DEBUG_PRINT("Installed AE Handlers");
    return ( err );
}

//
//
OSStatus do_Remove_AE_Handlers( void )
{
    OSStatus err = noErr;

    // err += do_Remove_AE_Handler( kCoreEventClass, kAEOpenApplication );
    // err += do_Remove_AE_Handler( kCoreEventClass, kAEReopenApplication );
    // err += do_Remove_AE_Handler( kCoreEventClass, kAEOpenDocuments );
    // err += do_Remove_AE_Handler( kCoreEventClass, kAEPrintDocuments );

    // err += do_Remove_AE_Handler( kCoreEventClass, kAEShowPreferences );
    // err += do_Remove_AE_Handler( kCoreEventClass, kAEQuitApplication );

    err += do_Remove_AE_Handler( typeWildcard, typeWildcard );

    DEBUG_PRINT("Removed AE Handlers");
    return ( err );
}
/*
//
OSStatus do_AE_DM_Notification( AppleEvent *event, AppleEvent *reply, long ref_con )
{
    DEBUG_PRINT("Got do_AE_DM_Notification() Apple Event");

    do_Handle_DM_Notification( event );

    return( errAEEve_NotHandled );
}

//
OSErr do_Handle_DM_Notification( AppleEvent *event )
{
    OSErr err = noErr;
    GrafPtr oldPort;
    AEDescList displayList, aDisplay;
    AERecord oldConfig, newConfig;
    AEKeyword tempWord;
    DisplayIDType displayID;
    unsigned long returnType;
    long count;
    Rect oldRect, newRect;

    GetPort(&oldPort);

    // Get a list of the displays from the Display Notice Apple event.
    err = AEGetParamDesc(event, kAEDisplayNotice, typeWildcard, &displayList);

    // How many items in the list?
    err = AECountItems(&displayList, &count);
    while (count > 0)
    {
        // Loop through the list.
        err = AEGetNthDesc(&displayList, count, typeWildcard, &tempWord, &aDisplay);

        // Get the old rect.
        err = AEGetNthDesc(&aDisplay, 1, typeWildcard, &tempWord, &oldConfig);
        err = AEGetPtrPtr(&oldConfig, keyDeviceRect, typeWildcard, &returnType, &oldRect, 8, nil);

        // Get the display ID so that we can get the GDevice later.
        err = AEGetPtrPtr(&oldConfig, keyDisplayID, typeWildcard, &returnType, &displayID, 8, nil);

        // Get the new rect.
        err = AEGetNthDesc(&aDisplay, 2, typeWildcard, &tempWord, &newConfig);
        err = AEGetPtrPtr(&newConfig, keyDeviceRect, typeWildcard, &returnType, &newRect, 8, nil);

        // If the new and old rects are not the same, we can assume that the GDevice has changed,
        // and the windows need to be rearranged.
        if( err == noErr && !EqualRect(&newRect, &oldRect) )
            HandleDeviceChange(displayID, &newRect);

        count--;
        err = AEDisposeDesc(&aDisplay);
    }
}
```

```
    err = AEDisposeDesc(&oldConfig);  
    err = AEDisposeDesc(&newConfig);  
}  
  
err = AEDisposeDesc(&displayList);  
SetPort(oldPort);  
  
return( err );  
}  
*/  
//-----  
  
OSStatus do_AE_Show_Preferences( AppleEvent *event, AppleEvent *reply, long ref_con )  
{  
    DEBUG_PRINT("Got do_AE_Show_Preferences() Apple Event");  
  
    return( errAEventNotHandled );  
}  
//-----  
  
OSStatus do_AE_Quit_App( AppleEvent *event, AppleEvent *reply, long ref_con )  
{  
    DEBUG_PRINT("Got do_AE_Quit_App() Apple Event");  
  
    do_Quit_App( (struct cal_globals *)ref_con );  
  
    return noErr;  
}  
//-----  
  
OSStatus do_AE_Not_Handled( AppleEvent *event, AppleEvent *reply, long ref_con )  
{  
    DEBUG_PRINT("Got do_AE_Not_Handled() Apple Event");  
  
    return( errAEventNotHandled );  
}
```

```
//
//                                     ©1998-2001 bergdesign inc.
//
//
#ifdef __o_cal_globals__
#define __o_cal_globals__
#endif

#ifdef __APPLE_CC__
#include <Carbon/Carbon.h>
#include <ApplicationServices/ApplicationServices.h>
#else
#if TARGET_API_MAC_CARBON
#include <Carbon.h>
#else
#include <CMCalibrator.h>
#include <Video.h>
#endif
#endif

#include <time.h>

// signatures
#define kMacCreatorCode 'ACal'
#define kIccManufacturerTag 'berg'
#define kIccPrivateTag 'berh'

//class o_vc_gamma;
class o_asst_dialog;
class o_response_window;
class o_black_level_window;
class o_white_point_window;
class o_black_window;
class o_viewer_window;

// These are the DITLs that can be brought up with
// the base assistant pane type. They have no
// custom user interaction.
enum
{
    kEndPaneDITL = 3600,

    kWhyDisplayTypePaneDITL = 2000,
    kWhyControlTypePaneDITL = 2050,
    kWhyBlackLevelPaneDITL = 2100,
    kCautionLCDBlackLevelPaneDITL = 3000,
    kWhyResponsePaneDITL = 2300,
    kWhyWhitePointPaneDITL = 2700,
    kWhyPhosphorTypePaneDITL = 2400,
    kCautionGenericPhosphorPaneDITL = 2500,
    kWhyTargetResponsePaneDITL = 2600,
    kCautionLCDResponsePaneDITL = 3000,

    kBaseAsstPaneAppendMode = -1
};

enum
{
    kProfileMenuID = 3800,
    kProfileMenuNewProfileItem = 1,
    kProfileMenuSeparatorItem = 2,
    kProfileMenuFirstProfileItem = 3
};

enum
{
    kDisplayTypeNone = 0,
    kDisplayTypeCRT = 1,
    kDisplayTypeLCD = 2,
    kDisplayTypeProjector = 3
};

enum
{
    kDisplayControlsNone = 0,
    kDisplayControlsBrightnessAndContrast = 1,
    kDisplayControlsBrightnessOnly = 2,
    kDisplayControlsContrastOnly = 3
};

enum
{
    kDisplayControlBlackLevel = 0,
    kDisplayControlPicture = 1
};
```

```
enum
{
    kPreferredPatternWidth      = 264,
    kPreferredPatternHeight     = 264,
    kCalibrationGradient        = 128,
    kCalibrationPattern_1x1     = 129,
    kCalibrationPattern_2x2     = 130,
    kCalibrationPattern_3x3     = 131,
    kCalibrationPattern_4x4     = 132,
    kCalibrationSolid_008       = 133,
    kCalibrationSolid_016       = 134,
    kCalibrationSolid_024       = 135,
    kCalibrationSolid_032       = 136,
    kCalibrationCenter          = 137,
    kCalibrationLines_1         = 150,
    kCalibrationLines_2         = 151
};

/* right now, neither of these are being checked -- i.e., they could be crashed */
#define MAX_CONTROL_POINTS 512 /* max number of points that can be calibrated */
#define MAX_TEST_POINTS 1024 /* number of test patterns when calibration is possible */

/* these set the the white point display scaling */
// #define MAX_WP_SAT 0.25
// #define MAX_WP_SAT 0.26
#define MAX_WP_SAT 1.0

/* these have to do with the iterative curve solutions */
#define ITMAX 100
#define EPS 3.0e-8
#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))
// #define MIN(a,b) ((a) < (b) ? (a) : (b))
// #define MAX(a,b) ((a) > (b) ? (a) : (b))

/* when this is 1, the pattern slider operates remapped through the y axis */
#define PAT_ON_Y 0

/* when this is 1, the pattern slider is remapped through the perceptual brightness */
#define PAT_PERCEPT 1

/* if this is defined the tests are done in black and white only */
// #define BW

// This is the difference between the pattern and the center in the black level measurement
// #define BBUMP(a) (pow((pow((a),2.5)+0.02),(1.0/2.5)))
#define BBUMP(a) ((a) * 1.05 + 0.05)

#define AMB_MAX 0.50
// #define AMB_MAX 1.0

#define FULL_X_TO_Y(x,cp)
    (get_y_from_x((x)*##cp##.##white_level,##cp##)*(1.0-##cp##.##black_level)+##cp##.##black_level)
#define FULL_X_TO_Y_PTR(x,cp)
    (get_y_from_x((x)*##cp##->##white_level,##cp##)*(1.0-##cp##->##black_level)+##cp##->##black_level)
#define FULL_X_TO_Y_PTR_NWP(x,cp) (get_y_from_x((x),##cp##)*(1.0-##cp##->##black_level)+##cp##->##black_level)

/* this is everything needed to do a properly scaled slider */
struct slider_info
{
    ControlHandle this_control;
    float current_value;
    float slider_min;
    float slider_max;
    float to_value_scale;
    float to_value_offset;
    float to_screen_scale;
    float to_screen_offset;
    float to_position_scale;
    float to_position_offset;
    WindowPtr my_window;
    Rect slider_rect;
    ProcPtr live_function;
    ControlActionUPP action_function;
    void *data_pointer;
};

/* pixel structures in video memory */

struct components
{
    unsigned char alpha;
```

```
    unsigned char red;
    unsigned char green;
    unsigned char blue;
};

struct f_color
{
    float red;
    float green;
    float blue;
};

/* information needed to draw the test pattern */
/* the light and dark colors used for test */
struct parent_info
{
    float a_x;
    float a_y;
    float b_x;
    float b_y;
};

/* pointers to arrays of calibrated points */
struct control_point_info
{
    int count; /* save - number of calibrated points */
    float black_level; /* save - lowest level that can be seen */
    float white_level; /* save - level to make the desired white */
    int *creation_order; /* represents what order the points were created in (used for undo) */
    float *x; /* save - input point */
    float *y; /* save - output point */
    float *shifted_x; /* input moved to nearest table entry in gamma table */
    float *y_for_shifted_x; /* the output at that point */
    float *indep; /* points along the 45 degree diagonal */
    float *dep; /* the deviation at that point along diagonal */
    float *d2; /* derivitaves used for smoothing */
    float *temp; /* temporary info used for smoothing */
    struct parent_info *parents; /* parents that were used to calibrate this point, */
    /* maintained but not used yet, but will be useful for undo */
};

/* scaling information between pixels, slider values, and floating point values */
struct scale_info
{
    Rect draw_rect;
    float x_min;
    float x_max;
    float y_min;
    float y_max;
    float to_screen_x_scale;
    float to_screen_x_offset;
    float to_screen_y_scale;
    float to_screen_y_offset;
    float to_value_x_scale;
    float to_value_x_offset;
    float to_value_y_scale;
    float to_value_y_offset;
};

/* floating point and pixel or slider value coordinates */
struct coordinates
{
    float fx;
    float fy;
    short sx;
    short sy;
};

/* used only for illustrator export */
struct ill_coord
{
    float fx_in;
    float fy_in;
    float fx_out;
    float fy_out;
};

/* information for a single test point (an array of these is used) */
/* to make math most efficient, test points are in an array of structures */
/* while control points are in a structure of arrays */
struct test_point_info
{
    float x; /* input point you'd be calibrating at */
    float y; /* estimate of output along curve at this point */
    int point_status; /* whether this test point has already been calibrated, */
};
```

```
    struct parent_info parents;      /* only true when both parents (control points) are the same */
};

#define INT_CONTROL_POINTS           MAX_CONTROL_POINTS * sizeof(int)
#define FLOAT_CONTROL_POINTS        MAX_CONTROL_POINTS * sizeof(float)
#define PARENT_INFO_CONTROL_POINTS MAX_CONTROL_POINTS * sizeof(struct parent_info)

// This will be slightly more than is needed, but will insure that we don't run out of space
#define kMaxChannelCount    3        // r, g & b
#define kMaxEntryCount      1024     // a typical table size is 256, but 10-bit entries need more
#define kMaxEntrySize       2        // 2 bytes for up to 16-bit entries
#define kMaxTableSize       ( kMaxChannelCount * kMaxEntryCount * kMaxEntrySize )
#define kMaxVideoCardGammaSize ( sizeof(CMVideoCardGamma) + sizeof(GammaTbl) + kMaxTableSize )

/* information about the graphics hardware */
struct graphics_dev_info
{
    CMDisplayIDType    display_id;

    int                channel_count;
    int                entry_count;
    int                entry_size;
    int                entry_size_bits;
    float              max_value;

#ifdef TARGET_API_MAC_CARBON
    CMVideoCardGamma   *gamma_table_w_header;
#else // TARGET_API_MAC_OS8
    GammaTbl           *gamma_table_w_header;
#endif
};

/* pointers and info that is necessary to calibrate a single color component */
struct calibration_component_info
{
    struct test_point_info    *test_points;
    int                       test_point_count;
    int                       nearest_point;

    struct control_point_info *cp_cur;
    int                       new_point;
    int                       new_cp_index;

    struct control_point_info *cp_r;
    struct control_point_info *cp_g;
    struct control_point_info *cp_b;

    struct scale_info         *wp_scale;
    struct scale_info         *plot_scale;
    float                     plot_target_gamma;
    Boolean                   plot_target_perceptual;

    struct f_color            component_color;

    struct graphics_dev_info  *saved_dev_info;
    struct graphics_dev_info  *this_dev_info;

    float                     max_pixel_value;
    int                       gam_tab_count;
    float                     gam_tab_max_value;

    struct slider_info        *pat_slider;
    struct slider_info        *adj_slider;
    struct slider_info        *amb_slider;
};

// we need to keep this in synch with the tri# resource and TMPL template
struct tri_chrom_values
{
    Fixed    red_X;
    Fixed    red_Y;
    Fixed    red_Z;
    Fixed    green_X;
    Fixed    green_Y;
    Fixed    green_Z;
    Fixed    blue_X;
    Fixed    blue_Y;
    Fixed    blue_Z;
    Fixed    black_X;
    Fixed    black_Y;
    Fixed    black_Z;
    long     white_count;
    long     native_white;
    Fixed    white_5000_X;
};
```

```
Fixed      white_5000_Y;
Fixed      white_5000_Z;
Fixed      white_6500_X;
Fixed      white_6500_Y;
Fixed      white_6500_Z;
Fixed      white_9000_X;
Fixed      white_9000_Y;
Fixed      white_9000_Z;
};

// we need to keep this in synch with the tri# resource and TMPL template
struct tri_info
{
    unsigned char    *manufacturer;
    unsigned char    *model;
    unsigned char    *variant;
    long             type;
    struct tri_chrom_values *values;
};

struct new_or_adjust_pane_text
{
    Str255    NewOrAdjustPaneText01;
    Str255    NewOrAdjustPaneText02;
    Str255    NewOrAdjustPaneText03;
    Str255    NewOrAdjustPaneText04;
    Str255    NewOrAdjustPaneText05;
    Str255    NewOrAdjustPaneText06;
    Str255    NewOrAdjustPaneText07;
    Str255    NewOrAdjustPaneText08;
    Str255    NewOrAdjustPaneText09;
};

struct cal_globals
{
    CalibrateEventUPP    event_proc;

    AVIDType             display_id;
    Rect                 display_bounds;

    CMPProfileRef        profile_ref;

    o_asst_dialog        *asst_dialog;
    Boolean               user_done;
    Boolean               create_profile;

    Boolean               expert_mode;    // user choice
    int                   number_of_channels; // probably redundant, but may hold unique value from expert_mode

    // new or adjust
    CMPProfileLocation    *profile_loc_array;
    int                   chosen_profile_index; // can be used to indicate whether profile is new or based on a
existing one
    CMPProfileLocation    chosen_profile_loc; // holds existing profile chosen by user

    SInt16                number_of_profiles;
    SInt16                count; // global counter used by iterator functions

    MenuRef               profile_menu;

    struct new_or_adjust_pane_text    pane_text;

    // display type
    short                 display_type;    // user choice of type
    Point                 display_resolution; // display manager
    int                   display_depth;    // display manager
    int                   display_frequency; // display manager

    // type of controls
    short                 controls_type;    // user choice of type
    int                   brightness_level; // user setting during measurement
    int                   contrast_level;   // user setting during measurement

    // adjust display
    o_black_window        *black_window;

    // black level
    o_black_level_window  *black_level_window;
    Boolean               black_level_complete; // progress

    // response
    o_response_window     *response_window;
    Boolean               response_complete;    // progress
    o_viewer_window       *viewer_window;
};
```



```
// white point
o_white_point_window    *white_point_window;
Boolean                  white_point_complete;    // progress

// chromaticities
Handle                   rez_data;
unsigned long            tri_count;
struct tri_info          *tri_data;              // tristimulus data
long                     tri_choice;             // user choice

// target gamma
float                     target_gamma;          // user choice
Boolean                  target_perceptual;      // user choice

// save profile
unsigned char             profile_name[256];
unsigned char             profile_file_name[32];

// date & time
struct tm                 date_time;              // date and time profile saved

struct calibration_component_info  this_component;
};

#endif /* __o_cal_globals__ */
```

```
#ifndef _COMPILEFLAGS_
#define _COMPILEFLAGS_

//-----
// Flags

// #define PRINT_DEBUG_TO_FILE
// #define PRINT_DEBUG_TO_STDIO

#define PRINT_STATUS_MESSAGES
#define PRINT_DEBUG_MESSAGES
#define PRINT_WARNING_MESSAGES
#define PRINT_ERROR_MESSAGES

// #define TARGET_CPU_PPC 1
// #define TARGET_OS_MAC 1

// #define TARGET_CARBON 1
// #define TARGET_API_MAC_OS8 0
// #define TARGET_API_MAC_CARBON 1
// #define TARGET_API_MAC_OSX 0

#define ACCESSOR_CALLS_ARE_FUNCTIONS 1
#define OPAQUE_TOOLBOX_STRUCTS 1
// #define OPAQUE_UPP_TYPES 1
// #define CALL_NOT_IN_CARBON 1

//-----
/*
Project settings:

PPC Target
1. Set the filename
2. Set the stack size, min and preferred heap size

C/C++ Language
1. Set the inline depth

Global Optimizations
1. Set level 3

PPC Processor
1. Turn on "Profiler Information"

PPC PEF
1. Set code sorting to "Use .arr file"
2. Profile the code and add/update the ".arr" file

Resources
Version number
1. Update the 'vers' resources
Copyright
1. Update the 'vers' resources
2. Update the 'ACal' resource
3. Update the 'plst' resource
4. Update the copyright info in the splash screen 'DLOG'
*/
//-----
#endif _COMPILEFLAGS_
```

```
#ifdef __APPLE_CC__
#include <Carbon/Carbon.h>
#else
#if TARGET_API_MAC_CARBON
#include <Carbon.h>
#else
#include <MacTypes.h>
#include <Memory.h>
#include <quickdraw.h>
#include <Movies.h>
#include <windows.h>
#include <video.h>
#include <Appearance.h>
#include <StandardFile.h>
#include <Script.h>
#include <Devices.h>
#include <Palettes.h>
#include <Resources.h>
#include <Displays.h>
#endif
#endif

#include <stdio.h>
#include <stdlib.h>
#include <cstdlib>
#include <string.h>
#include <math.h>

#include "globals.h"
#include "my_globals.h"
#include "my_strings.h"

int malloc_graphics_dev_info_vcg( struct graphics_dev_info * );
void free_graphics_dev_info_vcg( struct graphics_dev_info * );
void reset_graphics_dev_info( struct graphics_dev_info * );
int copy_graphics_dev_info( struct graphics_dev_info *, struct graphics_dev_info * );

int test_graphics_dev( struct graphics_dev_info * );

#if !TARGET_API_MAC_CARBON
int gamma_table_test( unsigned char *data_ptr, struct graphics_dev_info *this_dev_info );
#endif

#if TARGET_API_MAC_CARBON
UInt32 calc_gamma_size( CMVideoCardGamma * );
#else
UInt32 calc_gamma_size( GammaTbl * );
#endif

void copy_gamma_tables_to_dev_info( struct graphics_dev_info * );
void debug_dev_info( struct graphics_dev_info * );

int copy_gamma_from_dev( struct graphics_dev_info * );
int copy_gamma_to_dev( struct graphics_dev_info * );

int linear_gamma_to_dev( struct graphics_dev_info * );
int wacky_gamma_to_dev( struct graphics_dev_info * );

int make_linear_table( void *, UInt32, UInt32, UInt32 );
int make_wacky_table( void *, UInt32, UInt32, UInt32 );
```

```
#include "gamma_utils.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

//
// We'll malloc enough for the biggest possible table we could store here.
// It's relatively little memory in the grand scheme of things.
int malloc_graphics_dev_info_vcg(struct graphics_dev_info *this_dev_info)
{
    int err = noErr;

    // All the struct members will be initialized to 0 by calloc().

#ifdef TARGET_API_MAC_CARBON
    this_dev_info->gamma_table_w_header = (CMVideoCardGamma *)calloc( 1, kMaxVideoCardGammaSize );
#else
    this_dev_info->gamma_table_w_header = (GammaTbl *)calloc( 1, kMaxVideoCardGammaSize );
#endif

    if( NULL == this_dev_info->gamma_table_w_header )
        err = -1;

    DEBUG_VAR_PRINT("Passed thru malloc_graphics_dev_info_vcg() with error %d",err);
    return( err );
}

//
// This works for both Carbon and non-carbon calls
void free_graphics_dev_info_vcg(struct graphics_dev_info *this_dev_info)
{
    if( NULL != this_dev_info->gamma_table_w_header )
    {
        free( this_dev_info->gamma_table_w_header );
        this_dev_info->gamma_table_w_header = NULL;
    }

    DEBUG_PRINT("Passed thru free_graphics_dev_info_vcg()");
}

//
// This just initializes the local struct members.
// The members of the video card gamma table should be handled separately.
void reset_graphics_dev_info( struct graphics_dev_info *this_dev_info )
{
    this_dev_info->channel_count      = 0;
    this_dev_info->entry_count        = 0;
    this_dev_info->entry_size         = 0;
    this_dev_info->entry_size_bits    = 0;
    this_dev_info->max_value          = 0.0;
}

//
// This copies the entire graphics_dev_info structure, including any allocated gamma table memory.
int copy_graphics_dev_info( struct graphics_dev_info *dest_dev_info, struct graphics_dev_info *source_dev_info )
{
    int err = noErr;

    DEBUG_PRINT("Entered copy_graphics_dev_info()");

    if( source_dev_info != NULL && dest_dev_info != NULL )
    {
        // Copy the structure elements. This overwrites the crucial pointers, but we save them, then restore
        // them immediately after. This keeps us from having to do assignments on every single member.

        #if TARGET_API_MAC_CARBON
            CMVideoCardGamma *stored_ptr = dest_dev_info->gamma_table_w_header;
        #else
            GammaTbl *stored_ptr = dest_dev_info->gamma_table_w_header;
        #endif

        *dest_dev_info = *source_dev_info;
        dest_dev_info->gamma_table_w_header = stored_ptr;

        // copy the gamma table memory
        if( ( source_dev_info->gamma_table_w_header != NULL ) && ( dest_dev_info->gamma_table_w_header != NULL ) )
        {
            memcpy( dest_dev_info->gamma_table_w_header, source_dev_info->gamma_table_w_header,
                kMaxVideoCardGammaSize );
        }
        else if( ( source_dev_info->gamma_table_w_header == NULL ) && ( dest_dev_info->gamma_table_w_header == NULL ) )
        {
            // Everything is ok
        }
    }
}
```

```
    }
    else
    {
        err = paramErr;
    }
}
else
{
    err = paramErr;
}

DEBUG_VAR_PRINT("Left copy_graphics_dev_info() with error %d",err);
return( err );
}

//
#pragma mark -
//

int test_graphics_dev( struct graphics_dev_info *this_dev_info )
{
    int                err = noErr;
    struct graphics_dev_info    saved_dev_info;
    struct graphics_dev_info    test_dev_info;

    DEBUG_VAR_PRINT("Entered test_graphics_dev( %d )",this_dev_info->display_id);

    // Make sure it's a valid display id
    if( ( NULL == this_dev_info ) || ( this_dev_info->display_id == NULL ) )
    {
        err = paramErr;
        goto bail;
    }

    // Grab some memory for the gamma table we need to save
    err += malloc_graphics_dev_info_vcg( &saved_dev_info );
    if( err == noErr )
    {
        Boolean need_to_test = true;

        // Save the old table
        saved_dev_info.display_id = this_dev_info->display_id;
        err += copy_gamma_from_dev( &saved_dev_info );

/*
        // If we got a gamma table, we may already have what we need
        #if TARGET_API_MAC_CARBON
            if( saved_dev_info.gamma_table_w_header->tagType == cmVideoCardGammaTableType )
        #else
            if( saved_dev_info.gamma_table_w_header->gFormulaSize == 0 )
        #endif
        {
            DEBUG_PRINT("Found cmVideoCardGammaTableType already in the video card.");

            // We can pull the info we need from here
            copy_gamma_specs_to_dev_info( &saved_dev_info );

            // Can a color card ever have a monochrome gamma table? Yes it can.
            // The sRGB Profile loads one under Mac OS X, so we need to account for it.
            if( saved_dev_info.channel_count == 3 )
            {
                DEBUG_PRINT("cmVideoCardGammaTableType had 3 channels. Done.");
                copy_graphics_dev_info( this_dev_info, &saved_dev_info );
                need_to_test = false;
            }
            else
            {
                DEBUG_PRINT("cmVideoCardGammaTableType did not have 3 channels.");
            }
        }
*/
        if( need_to_test ) // Otherwise, we still need to test the card
        {
            DEBUG_PRINT("Found something other than cmVideoCardGammaTableType in the video card.");
            DEBUG_PRINT("Found something other than cmVideoCardGammaTableType in the video card, or we had to test the card anyway.");
            DEBUG_PRINT(" Now trying to determine what the video card can hold...");

            // Grab some memory for the test gamma table
            err += malloc_graphics_dev_info_vcg( &test_dev_info );
            if( err == noErr )
            {
                // We push a wacky table to the device so we can check the result
                // and see what ColorSync was successfully able to put there.
                test_dev_info.display_id = this_dev_info->display_id;
                test_dev_info.channel_count = 3;
                test_dev_info.entry_count = 256;
            }
        }
    }
}
```

```
test_dev_info.entry_size = 1;
err += wacky_gamma_to_dev( &test_dev_info );
if( err == noErr )
{
    // Retrieve the resulting table
    err += copy_gamma_from_dev( &test_dev_info );
    if( err == noErr )
    {
#if TARGET_API_MAC_CARBON
        if( test_dev_info.gamma_table_w_header->tagType == cmVideoCardGammaTableType )
#else
        if( test_dev_info.gamma_table_w_header->gFormulaSize == 0 )
#endif
        {
            DEBUG_PRINT("Video card could hold cmVideoCardGammaTableType and return it.");

            // can pull the info we need from here
            copy_gamma_specs_to_dev_info( &test_dev_info );
            copy_graphics_dev_info( this_dev_info, &test_dev_info );

            // initialize the gamma table
#if TARGET_API_MAC_CARBON
            make_linear_table( this_dev_info->gamma_table_w_header->u.table.data,
                              this_dev_info->gamma_table_w_header->u.table.channels,
                              this_dev_info->gamma_table_w_header->u.table.entryCount,
                              this_dev_info->gamma_table_w_header->u.table.entrySize );
#else
            make_linear_table( this_dev_info->gamma_table_w_header->gFormulaData,
                              this_dev_info->gamma_table_w_header->gChanCnt,
                              this_dev_info->gamma_table_w_header->gDataCnt,
                              BITS2BYTES(this_dev_info->gamma_table_w_header->gDataWidth) );
#endif
        }
        else
        {
            DEBUG_PRINT("Video card returned gamma formulas or an unknown gamma type.");
            err += -2;
        }
    }

    free_graphics_dev_info_vcg( &test_dev_info );

    // Push the saved table/formula back to the card
    err += copy_gamma_to_dev( &saved_dev_info );
}

free_graphics_dev_info_vcg( &saved_dev_info );
}

bail:
    DEBUG_VAR_PRINT("Left test_graphics_dev() with error %d",err);
    return( err );
}

/*
int test_graphics_dev( struct graphics_dev_info *this_dev_info )
{
    int err = noErr;
    //struct graphics_dev_info saved_dev_info;
    //struct graphics_dev_info test_dev_info;

    // make sure it is a valid display id
    if( this_dev_info->display_id == NULL )
        return( paraErr );

    // we'll start assuming that it's in color
    this_dev_info->gamma_table_w_header->gChanCnt = 3;
    this_dev_info->channel_count = 3;

    // make the structure match the hardware before testing
    err = copy_gamma_from_dev(this_dev_info);
    if(err)
        return( -4 );

    copy_gamma_specs_to_dev_info( this_dev_info );

    wacky_gamma_to_dev( this_dev_info );

    // initialize some pointers into the data
    UInt32 offset = this_dev_info->entry_count * this_dev_info->entry_size;
    unsigned char *red_ptr = (unsigned char *)this_dev_info->gamma_table_w_header->gFormulaData;
    unsigned char *blue_ptr = red_ptr + offset;
    unsigned char *green_ptr = green_ptr + offset;
}
```

```
// check red
err = gamma_table_test( red_ptr, this_dev_info );
if(err)
    return( err );

// now we'll check to make sure that it can really handle color
err = gamma_table_test( green_ptr, this_dev_info );
if(err)
{
    // wasn't really in color, we'll just change the structure
    this_dev_info->gamma_table_w_header->gChanCnt = 1;
    this_dev_info->channel_count = 1;
}
else
{
    err = gamma_table_test( blue_ptr, this_dev_info );
    if(err)
    {
        // wasn't really in color, we'll just change the structure
        this_dev_info->gamma_table_w_header->gChanCnt = 1;
        this_dev_info->channel_count = 1;
    }
}

return( err );
}
*/
//
//
#if !TARGET_API_MAC_CARBON

int gamma_table_test( unsigned char *data_ptr, struct graphics_dev_info *this_dev_info )
{
    unsigned char old_uchar, trial_uchar;
    unsigned short old_ushort, trial_ushort;
    int err = noErr;

    /* we won't really know if this works until we get to test with a card that is >8 bits */

    /* check if <=8 bits */
    if(this_dev_info->entry_size == 1)
    {
        /* save what was there so we can put it back */
        old_uchar = *(data_ptr + (this_dev_info->entry_count - 1));

        /* put max - 1 into the top location for this color (avoids too much flicker during change) */
        *(data_ptr + (this_dev_info->entry_count - 1)) = trial_uchar = (unsigned char)this_dev_info->max_value - 1;

        /* send it to the card */
        err = copy_gamma_to_dev(this_dev_info);
        if(err)
            return( -11 );

        /* read it back from the card */
        err = copy_gamma_from_dev(this_dev_info);
        if(err)
            return( -12 );

        /* see if they matched */
        if(*(data_ptr + (this_dev_info->entry_count - 1)) != trial_uchar)
            return( -13 );

        /* put the max value into the top location for this color */
        *(data_ptr + (this_dev_info->entry_count - 1)) = trial_uchar = (unsigned char)this_dev_info->max_value;

        /* send it to the card */
        err = copy_gamma_to_dev(this_dev_info);
        if(err)
            return( -14 );

        /* read it back from the card */
        err = copy_gamma_from_dev(this_dev_info);
        if(err)
            return( -15 );

        /* see if they matched */
        if(*(data_ptr + (this_dev_info->entry_count - 1)) != trial_uchar)
            return( -16 );

        /* restore the original value */
        *(data_ptr + (this_dev_info->entry_count - 1)) = old_uchar;

        err = copy_gamma_to_dev(this_dev_info);
        if(err)
            return( -17 );
    }
    else

```

```
{
    /* save what was there so we can put it back */
    old_ushort = *((unsigned short *)data_ptr + (this_dev_info->entry_count - 1));

    /* put max - 1 into the top location for this color */
    *((unsigned short *)data_ptr + (this_dev_info->entry_count - 1)) = trial_ushort = (unsigned
short)this_dev_info->max_value - 1;

    /* send it to the card */
    err = copy_gamma_to_dev(this_dev_info);
    if(err)
        return( -11 );

    /* read it back from the card */
    err = copy_gamma_from_dev(this_dev_info);
    if(err)
        return( -12 );

    /* see if they matched */
    if(*((unsigned short *)data_ptr + (this_dev_info->entry_count - 1)) != trial_ushort)
        return( -13 );

    /* put the max value into the top location for this color */
    *((unsigned short *)data_ptr + (this_dev_info->entry_count - 1)) = trial_ushort = (unsigned
short)this_dev_info->max_value;

    /* send it to the card */
    err = copy_gamma_to_dev(this_dev_info);
    if(err)
        return( -14 );

    /* read it back from the card */
    err = copy_gamma_from_dev(this_dev_info);
    if(err)
        return( -15 );

    /* see if they matched */
    if(*((unsigned short *)data_ptr + (this_dev_info->entry_count - 1)) != trial_ushort)
        return( -16 );

    /* restore the original value */
    *((unsigned short *)data_ptr + (this_dev_info->entry_count - 1)) = old_ushort;

    err = copy_gamma_to_dev(this_dev_info);
    if(err)
        return( -17 );
}

return 0;
}

#endif

//
#pragma mark -
//

#if TARGET_API_MAC_CARBON

UInt32 calc_gamma_size( CMVideoCardGamma *gamma_table_w_header )
{
    if( ( NULL != gamma_table_w_header ) && ( gamma_table_w_header->tagType == cmVideoCardGammaTableType ) )
    {
        // This will get the exact size of the new CMVideoCardGamma struct with a CMVideoCardGammaTable in it.
        // Since "data[1]" in the CMVideoCardGammaTable struct is only a char, there is some padding for 68k data
        alignment.
        UInt32 header_size = sizeof(CMVideoCardGamma)
            - MAX( sizeof(CMVideoCardGammaTable), sizeof(CMVideoCardGammaFormula) )
            + sizeof(CMVideoCardGammaTable)
            - sizeof(short); // padding for data[1]

        UInt32 formula_size = 0;
        UInt32 data_size = gamma_table_w_header->u.table.channels *
            gamma_table_w_header->u.table.entryCount *
            gamma_table_w_header->u.table.entrySize;

        UInt32 gtw_h_size = header_size + formula_size + data_size;

        // A little reality check in case the pointer is not NULL and the data is invalid.
        DEBUG_VAR_PRINT( "CMVideoCardGammaSize: %d", kMaxVideoCardGammaSize);
        DEBUG_VAR_PRINT( "header_size: %d", header_size );
        DEBUG_VAR_PRINT( "formula_size: %d", formula_size );
        DEBUG_VAR_PRINT( "data_size: %d", data_size );
        DEBUG_VAR_PRINT( "gtw_h_size: %d", gtw_h_size);

        if( gtw_h_size <= kMaxVideoCardGammaSize )
            return( gtw_h_size );
    }
}
```



```
        else
            return( 0 );
    }
    else
    {
        return( 0 );
    }
}

#else

UInt32 calc_gamma_size( GammaTbl *gamma_table_w_header )
{
    if( NULL != gamma_table_w_header )
    {
        // This will get the exact size of the old QuickDraw GammaTbl
        UInt32 header_size = sizeof( GammaTbl ) - sizeof( short );
        UInt32 formula_size = gamma_table_w_header->gFormulaSize;
        UInt32 data_size = gamma_table_w_header->gChanCnt *
            gamma_table_w_header->gDataCnt *
            BITS2BYTES( gamma_table_w_header->gDataWidth );

        UInt32 gtw_h_size = header_size + formula_size + data_size;

        // A little reality check in case the pointer is not NULL and the data is invalid.
        DEBUG_VAR_PRINT("kMaxVideoCardGammaSize: %d",kMaxVideoCardGammaSize);
        DEBUG_VAR_PRINT("header_size: %d", header_size );
        DEBUG_VAR_PRINT("formula_size: %d", formula_size );
        DEBUG_VAR_PRINT("data_size: %d", data_size );
        DEBUG_VAR_PRINT("gtw_h_size: %d",gtw_h_size);

        if( gtw_h_size <= kMaxVideoCardGammaSize )
            return( gtw_h_size );
        else
            return( 0 );
    }
    else
    {
        return( 0 );
    }
}

#endif

//
// This supplements copy_gamma_from_dev(). If you want the graphics_dev_info members to
// match the gamma table, call this function after a call to copy_gamma_from_dev().

void copy_gamma_specs_to_dev_info( struct graphics_dev_info *this_dev_info )
{
    #if TARGET_API_MAC_CARBON
    {
        if( this_dev_info->gamma_table_w_header->tagType == cmVideoCardGammaTableType )
        {
            this_dev_info->channel_count = this_dev_info->gamma_table_w_header->u.table.channels;
            this_dev_info->entry_count = this_dev_info->gamma_table_w_header->u.table.entryCount;
            this_dev_info->entry_size = 1; // force this for now

/*
            // Our hack to work around OS 9 bug returning 2 bytes per entry in some gamma tables
            if( do_Check_If_Running_On_Carbon_X )
            {
                DEBUG_PRINT("Running on OS X... Questioning values returned from test gamma table...");
                this_dev_info->entry_size = this_dev_info->gamma_table_w_header->u.table.entrySize;

                // ++++++ remove this ++++++
                this_dev_info->entry_size = 2;

                if( this_dev_info->entry_size > 1 )
                {
                    Str255 error_msg;
                    error_msg[0] = 0;
                    do_p_strcat(error_msg,this_dev_info->entry_size);
                    do_p_strcat(error_msg,"p byte(s) per entry is very uncommon and may be the result of a device
driver bug. If you click \"Yes\" and the measurement screens do not respond correctly, relaunch SuperCal and
choose \"No\".");

                    // Ask the user if they want to trust the system or not.
                    NSInteger answer = kAlertStdAlertCancelButton;
                    NSInteger r = do_Two_Button_Alert ( kAlertCautionAlert,
                        "\pDo you want to use the gamma table entry size reported by yo
card?",
                        error_msg,
                        "\pYes",
                        "\pNo" );

                    DEBUG_VAR_PRINT("User choice: %d",answer);
                    if( kAlertStdAlertCancelButton == answer )
                        this_dev_info->entry_size = 1;
                }
            }
*/
        }
    }
    #endif
}
```

```
    }  
    else  
    {  
        DEBUG_PRINT("Running on pre OS X... Forcing 1 byte per entry...");  
        this_dev_info->entry_size = 1;  
    }  
*/  
    this_dev_info->entry_size_bits = this_dev_info->entry_size * 8;  
    this_dev_info->max_value = (float)(( 1 << this_dev_info->entry_size_bits ) - 1 );  
} else  
{  
    reset_graphics_dev_info( this_dev_info );  
}  
}  
#endif  
  
    this_dev_info->channel_count = this_dev_info->gamma_table_w_header->gChanCnt;  
    this_dev_info->entry_count = this_dev_info->gamma_table_w_header->gDataCnt;  
    this_dev_info->entry_size_bits = this_dev_info->gamma_table_w_header->gDataWidth;  
    this_dev_info->entry_size = BITS2BYTES( this_dev_info->entry_size_bits );  
    this_dev_info->max_value = (float)(( 1 << this_dev_info->entry_size_bits ) - 1 );  
}  
#endif  
  
    debug_dev_info( this_dev_info );  
    DEBUG_PRINT("Passed thru copy_gamma_specs_to_dev_info()");  
}  
  
//  
  
void debug_dev_info( struct graphics_dev_info *this_dev_info )  
{  
    DEBUG_VAR_PRINT("display_id: %d", this_dev_info->display_id);  
    DEBUG_VAR_PRINT("channel_count: %d", this_dev_info->channel_count);  
    DEBUG_VAR_PRINT("entry_count: %d", this_dev_info->entry_count);  
    DEBUG_VAR_PRINT("entry_size: %d", this_dev_info->entry_size);  
    DEBUG_VAR_PRINT("entry_size_bits: %d", this_dev_info->entry_size_bits);  
    DEBUG_VAR_PRINT("max_value: %f", this_dev_info->max_value);  
}  
  
//  
// copy_gamma_from_dev() copies the gamma information from a device into  
// the pre-allocated space in a graphics_dev_info struct. It will overwrite  
// any gamma information already contained in the graphics_dev_info struct.  
//  
// It does not synchronize the members in the graphics_dev_info struct with  
// the members of the gamma table. This way, you can get the card data for  
// test purposes without polluting the graphics_dev_info struct members.  
  
int copy_gamma_from_dev(struct graphics_dev_info *this_dev_info )  
{  
    int err = noErr;  
  
    DEBUG_VAR_PRINT("Entered copy_gamma_from_dev( %d )",this_dev_info->display_id);  
  
    // make sure it is a valid display id  
    if( ( this_dev_info->display_id != NULL ) && ( this_dev_info->gamma_table_w_header != NULL ) )  
    {  
        #if TARGET_API_MAC_CARBON  
        {  
            // Under Carbon, we have to use the ColorSync 3.0+ calls to get to the video card.  
            UInt32 vcg_size = 0;  
            err = CMGetGammaByAVID( this_dev_info->display_id, NULL, &vcg_size );  
            if( noErr == err )  
            {  
                err = CMGetGammaByAVID( this_dev_info->display_id, this_dev_info->gamma_table_w_header, &vcg_size )  
            }  
        }  
        #else  
        {  
            // Under InterfaceLib, we use the old PBStatus calls to get to the video card.  
            GDHandle device;  
            err = GDGetDeviceByDisplayID ( this_dev_info->display_id, &device, false );  
            if( noErr == err )  
            {  
                VDGammaRecord myVDGammaRecord;  
                ControlParam control;  
  
                myVDGammaRecord.csGTable = NULL;  
  
                control.ioCompletion = NULL;  
                control.ioCRefNum = (**device).gdRefNum;  
                control.csCode = cscGetGamma;  
                *((Ptr *)&control.csParam[0]) = (Ptr)&myVDGammaRecord;  
  
                err = PBStatusSync((ParmBlkPtr) &control);  
            }  
        }  
    }  
}
```

```
        if( err == noErr )
        {
            GammaTbl *card_gamma_tbl = (GammaTbl *)myVDGammaRecord.csGTable;

            UInt32 gtw_h_size = calc_gamma_size( card_gamma_tbl );

            if( gtw_h_size > 0 )
                memcpy( this_dev_info->gamma_table_w_header, card_gamma_tbl, gtw_h_size );
            else
                err = -1;
        }
    }
}
#endif
}
else
{
    err = paramErr;
}

DEBUG_VAR_PRINT("Left copy_gamma_from_dev() with error %d",err);
return( err );
}

//
// copy_gamma_to_dev() copies the gamma information from a graphics_dev_info struct
// to a display device.
int copy_gamma_to_dev( struct graphics_dev_info *this_dev_info )
{
    int err = noErr;

    DEBUG_VAR_PRINT("Entered copy_gamma_to_dev( %d )",this_dev_info->display_id);

    // make sure it is a valid display id
    if( ( NULL != this_dev_info ) && ( NULL != this_dev_info->display_id ) && ( NULL !=
this_dev_info->gamma_table_w_header ) )
    {
        #if TARGET_API_MAC_CARBON
        {
            err = CMSetGammaByAVID( this_dev_info->display_id, this_dev_info->gamma_table_w_header );
        }
        #else
        {
            GDHandle device;
            err = DMGetGDeviceByDisplayID ( this_dev_info->display_id, &device, false );
            if( noErr == err )
            {
                VDDGammaRecord myVDGammaRecord;
                ControlParam control;

                myVDGammaRecord.csGTable = (Ptr)this_dev_info->gamma_table_w_header;

                control.ioCompletion = NULL;
                control.ioCRefNum = (**device).gdRefNum;
                control.csCode = cscSetGamma;
                *((Ptr *)&control.csParam[0]) = (Ptr)&myVDGammaRecord;

                err = PBControlSync( (ParamBlkPtr)&control );
            }
        }
        #endif
    }
    else
    {
        err = paramErr;
    }

    DEBUG_VAR_PRINT("Left copy_gamma_to_dev() with error %d",err);
    return( err );
}

//
#pragma mark -

//
// This function pushes a linear gamma table to the video card.
// Be careful if you expect the same data to be returned after
// you push this to the card. ColorSync watches and reduces the
// data to the minimum that is required to get the same effect.
// For example, if you push 3 channels that are all the same,
// ColorSync will reduce it to one channel for you.
// Nice of them, isn't it?

int linear_gamma_to_dev(struct graphics_dev_info *this_dev_info)
{
    int err = noErr;
```

```
DEBUG_VAR_PRINT("Entered linear_gamma_to_dev( %d )",this_dev_info->display_id);
if( ( this_dev_info != NULL ) && ( this_dev_info->display_id != NULL ) )
{
    #if TARGET_IPI_MAC_CARBON
    {
        CMVideoCardGamma *gtwh = (CMVideoCardGamma *)calloc( 1, sizeof(CMVideoCardGamma) );
        if( NULL != gtwh )
        {
            gtwh->tagType          = cmVideoCardGammaTableType;
            gtwh->u.table.channels  = this_dev_info->channel_count;
            gtwh->u.table.entryCount = this_dev_info->entry_count;
            gtwh->u.table.entrySize = this_dev_info->entry_size;

            UInt32 gtwh_size = calc_gamma_size( gtwh );
            if( 0 != gtwh_size )
            {
                CMVideoCardGamma *new_gtwh = (CMVideoCardGamma *)realloc( gtwh, gtwh_size );
                if( NULL != new_gtwh )
                {
                    make_linear_table( new_gtwh->u.table.data,
                                       new_gtwh->u.table.channels,
                                       new_gtwh->u.table.entryCount,
                                       new_gtwh->u.table.entrySize );

                    err = CMSetGammaByAVID( this_dev_info->display_id, new_gtwh );

                    free( new_gtwh );
                }
                else
                {
                    err = memFullErr;
                    free( gtwh );
                }
            }
            else
            {
                err = paramErr;
                free( gtwh );
            }
        }
    }
    #else
    {
        GDHandle device;
        err = DMGetGDDeviceByDisplayID( this_dev_info->display_id, &device, false );
        if( noErr == err )
        {
            VDGammaRecord myVDGammaRecord;
            CntrlParam control;

            myVDGammaRecord.csGTable      = NULL; // Make the data pointer NULL and the card will linearize it
            control.ioCompletion          = NULL;
            control.ioCRefNum             = (**device).gdRefNum;
            control.csCode                 = cscSetGamma;
            *((Ptr *)&control.csParam[0]) = (Ptr)&myVDGammaRecord;

            err = PBControlSync( (ParmBlkPtr) &control );
        }
    }
    #endif
}
else
{
    err = paramErr;
}

DEBUG_VAR_PRINT("Left linear_gamma_to_dev() with error %d",err);
return( err );
}

//
// We use this function to get around ColorSync's uncanny ability to read
// our minds and reduce what we are pushing to the card to its bare minimum.
// By creating multiple channels with no data correlation, ColorSync can't
// reduce it, so we can check it after we send it and see what the card
// managed to hold.
//
// We should try pushing a big honking 2 byte, umpteen entry table
// to the card and see what ColorSync does to insure it gets there,
// or what it does when it fails.
// We'll just do a 1 byte entry for now to be safe.
//
/*
int wacky_gamma_to_dev(struct graphics_dev_info *this_dev_info)
{

```

```
int      err = noErr;

DEBUG_PRINT("Entered wacky_gamma_to_dev()");

if( ( this_dev_info != NULL ) && ( this_dev_info->display_id != NULL ) )
{
    CMVideoCardGamma temp_gtwh;

    temp_gtwh.tagType          = cmVideoCardGammaTableType;
    temp_gtwh.u.table.channels = this_dev_info->channel_count;
    temp_gtwh.u.table.entryCount = this_dev_info->entry_count;
    temp_gtwh.u.table.entrySize = this_dev_info->entry_size;

    UInt32 gtwh_size = calc_gamma_size( &temp_gtwh );
    if( 0 != gtwh_size )
    {
        CMVideoCardGamma *gtwh = (CMVideoCardGamma *)calloc( 1, gtwh_size );
        if( NULL != gtwh )
        {
            *gtwh = temp_gtwh;

            DEBUG_VAR_PRINT("CMVideoCardGamma* (0x%X)", gtwh );
            DEBUG_EXT_VAR_PRINT(" to (0x%X)", ((char *)gtwh + gtwh_size));
            DEBUG_VAR_PRINT("gtwh->u.table.data (0x%X)", gtwh->u.table.data );

            make_wacky_table(    gtwh->u.table.data,
                                gtwh->u.table.channels,
                                gtwh->u.table.entryCount,
                                gtwh->u.table.entrySize );

            err = CMSSetGammaByAVID( this_dev_info->display_id, gtwh );

            free( gtwh );
        }
        else
        {
            err = memFullErr;
        }
    }
    else
    {
        err = paramErr;
    }
}
else
{
    err = paramErr;
}

DEBUG_VAR_PRINT("Left wacky_gamma_to_dev() with error %d",err);
return( err );
}
*/

int wacky_gamma_to_dev(struct graphics_dev_info *this_dev_info)
{
    int      err = noErr;

    DEBUG_VAR_PRINT("Entered wacky_gamma_to_dev( %d )",this_dev_info->display_id);

    if( ( this_dev_info != NULL ) && ( this_dev_info->display_id != NULL ) )
    {
        #if TARGET_API_MAC_CARBON
        {
            CMVideoCardGamma *gtwh = (CMVideoCardGamma *)calloc( 1, sizeof(CMVideoCardGamma) );
            if( NULL != gtwh )
            {
                gtwh->tagType          = cmVideoCardGammaTableType;
                gtwh->u.table.channels = this_dev_info->channel_count;
                gtwh->u.table.entryCount = this_dev_info->entry_count;
                gtwh->u.table.entrySize = this_dev_info->entry_size;

                UInt32 gtwh_size = calc_gamma_size( gtwh );
                if( 0 != gtwh_size )
                {
                    CMVideoCardGamma *new_gtwh = (CMVideoCardGamma *)realloc( gtwh, gtwh_size );
                    if( NULL != new_gtwh )
                    {
                        make_wacky_table(    new_gtwh->u.table.data,
                                            new_gtwh->u.table.channels,
                                            new_gtwh->u.table.entryCount,
                                            new_gtwh->u.table.entrySize );

                        err = CMSSetGammaByAVID( this_dev_info->display_id, new_gtwh );

                        free( new_gtwh );
                    }
                }
            }
        }
        #endif
    }
}
```

```

    {
        err = memFullErr;
        free( gtwh );
    }
    else
    {
        err = paramErr;
        free( gtwh );
    }
}
}
#else
{
    GammaTbl *gtwh = (GammaTbl *)calloc( 1, sizeof(GammaTbl) );
    if( NULL != gtwh )
    {
        gtwh->gVersion      = 0;
        gtwh->gType         = 0;
        gtwh->gFormulaSize  = 0;
        gtwh->gChanCnt      = this_dev_info->channel_count;
        gtwh->gDataCnt       = this_dev_info->entry_count;
        gtwh->gDataWidth    = this_dev_info->entry_size * 8; // bytes to bits

        UInt32 gtwh_size = calc_gamma_size( gtwh );
        if( 0 != gtwh_size )
        {
            GammaTbl *new_gtwh = (GammaTbl *)realloc( gtwh, gtwh_size );
            if( NULL != new_gtwh )
            {
                make_wacky_table( new_gtwh->gFormulaData,
                                new_gtwh->gChanCnt,
                                new_gtwh->gDataCnt,
                                BITS2BYTES(new_gtwh->gDataWidth) );

                struct graphics_dev_info temp_dev_info;
                temp_dev_info.display_id = this_dev_info->display_id;
                temp_dev_info.gamma_table_w_header = new_gtwh;

                err = copy_gamma_to_dev( &temp_dev_info );

                free( new_gtwh );
            }
            else
            {
                err = memFullErr;
                free( gtwh );
            }
        }
        else
        {
            err = paramErr;
            free( gtwh );
        }
    }
}
#endif
}
else
{
    err = paramErr;
}

DEBUG_VAR_PRINT("Left wacky_gamma_to_dev() with error %d",err);
return( err );
}

//
#pragma mark -
//

int make_linear_table( void *data, UInt32 channel_count, UInt32 entry_count, UInt32 entry_size )
{
    int err = noErr;

    if( ( ( channel_count * entry_count * entry_size ) > kMaxTableSize ) || ( NULL == data ) )
    {
        err = paramErr;
    }
    else
    {
        UInt32 max_value = ( 1 << (entry_size * 8) ) - 1;
        float multiplier = ( 1 << (entry_size * 8) ) / entry_count;

        for( int i=0; i < channel_count; i++ )
        {

```

```
        for( int j=0; j < entry_count; j++ )
        {
            if( 1 == entry_size )
                ((UInt32 *)data)[ j + ( i * entry_count ) ] = (UInt8)(( j + 1 ) * multiplier ) - 1 );
            else
                ((UInt16 *)data)[ j + ( i * entry_count ) ] = (UInt16)(( j + 1 ) * multiplier ) - 1 );
        }
    }

    DEBUG_VAR_PRINT("Passed thru make_linear_table() with error %d",err);
    return( err );
}

//
int make_wacky_table( void *data, UInt32 channel_count, UInt32 entry_count, UInt32 entry_size )
{
    int err = noErr;

    if( ( ( channel_count * entry_count * entry_size ) > kMaxTableSize ) || ( NULL == data ) )
    {
        err = paramErr;
    }
    else
    {
        UInt32 max_value = ( 1 << (entry_size * 8) ) - 1;
        UInt8 k = 0;

        //
        DEBUG_VAR_PRINT("gamma table (0x%X)", data );
        //
        DEBUG_PRINT("(channel,entry):");

        for( int i=0; i < channel_count; i++ )
        {
            for( int j=0; j < entry_count; j++ )
            {
                if( i == 0 )
                {
                    //
                    k = j;
                    k = pow( ( j / (float)entry_count ) , 1.0/1.8 ) * max_value;
                }
                else if( i == 1 )
                {
                    //
                    k = max_value - j;
                    k = pow( ( j / (float)entry_count ) , 1.0/1.9 ) * max_value;
                }
                else
                {
                    //
                    k = j/2;
                    k = pow( ( j / (float)entry_count ) , 1.0/2.0 ) * max_value;
                }

                //
                DEBUG_VAR_PRINT("(%d",i);
                //
                DEBUG_EXTRA_VAR_PRINT(",%d) = ",j);
                //
                DEBUG_EXTRA_VAR_PRINT("%d",k);
                //
                DEBUG_EXTRA_VAR_PRINT(" (0x%X)", &(data[ j + ( i * entry_count ) ]));

                if( 1 == entry_size )
                {
                    ((UInt8 *)data)[ j + ( i * entry_count ) ] = (UInt8)k;
                }
                else
                {
                    ((UInt16 *)data)[ j + ( i * entry_count ) ] = (UInt16)k;
                }
            }
        }

        DEBUG_VAR_PRINT("Passed thru make_wacky_table() with error %d",err);
        return( err );
    }
}

/*
{
int err = noErr;

if( ( ( channel_count * entry_count * entry_size ) > kMaxTableSize ) || ( NULL == data ) )
{
    err = paramErr;
}
else
{
    float max_value = (float)(( 1 << (entry_size * 8) ) - 1 );
    float inc = 1 / ( max_value / 3.1415926536 );
    UInt16 k;

    for( int i=0; i < channel_count; i++ )
    {

```

```
for( int i=0; j < entry_count; j++ )
{
    // This is a real hack
    if( i == 0 )
        k = j;
    else if( i == 1 )
        k = max_value - j; // inverse ramp
    else if( i == 2 )
        k = sin( j * incr ) * max_value; // sin wave

    if( i == entry_size )
        ((UInt8 *)data)[ j + ( i * entry_count ) ] = (UInt8)k;
    else
        ((UInt16 *)data)[ j + ( i * entry_count ) ] = (UInt16)k;
}
}

DEBUG_VAR_PRINT("Passed thru make_wacky_table() with error %d",err);
return( err );
}
*/
```



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "globals.h"
#include "my_macros.h"
#include "my_colors.h"
#include "gamma_utils.h"

#ifdef __APPLE_CC__
#include <Carbon/Carbon.h>
#else
#ifdef TARGET_API_MAC_CARBON
#include <Carbon.h>
#else
#include <MacTypes.h>
#include <Memory.h>
#include <quickdraw.h>
#include <Movies.h>
#include <windows.h>
#include <video.h>
#include <Appearance.h>
#include <ControlDefinitions.h>
#include <StandardFile.h>
#include <Script.h>
#include <Devices.h>
#include <Palettes.h>
#include <Resources.h>
#endif
#endif
#endif

void initialize_control_point_ptrs( struct control_point_info * );
int get_control_point_memory(struct control_point_info *);
void dump_control_point_memory(struct control_point_info *);
int stream_out_control_point_info( char *, struct control_point_info * );
int stream_in_control_point_info( struct control_point_info *, char * );
UInt32 get_control_point_info_size( int );
void print_control_point_info( struct control_point_info * );

void initialize_slider(struct slider_info *);
void dispose_slider(struct slider_info *);

void initialize_slider_scale(struct slider_info *);
float slider_to_value(short ,struct slider_info *);
short value_to_screen(float ,struct slider_info *);
void force_slider_position(float ,struct slider_info *);

void initialize_palette(WindowPtr,PaletteHandle *);

int update_colors(struct graphics_dev_info *,struct components *,int);
void value_to_comp_color(float , struct f_color *, struct calibration_component_info *);
void offset_scale_f_pixel(struct f_color *, struct calibration_component_info *);
void color_float_to_pixel(struct f_color *, struct components *,struct calibration_component_info *);

void reset_control_points(struct control_point_info *);
void calculate_smoothing(struct control_point_info *);
void make_shifted_control_points(struct calibration_component_info *);
void locate_new_test_points(struct calibration_component_info *);
float locate_value(float (*func)(float,float,struct control_point_info *), float x1, float x2, float tol,float
target,struct control_point_info *my_control_points);
int compare_points(const void *,const void *);
void set_adj_scale(struct calibration_component_info *);

float quantitize_tab_position(float,struct calibration_component_info *);
float get_y_from_x(float,void *);
float param_from_x(float,float,struct control_point_info *);

float percept_to_lum(float);
float lum_to_percept(float);

float interpolate_value(float,struct control_point_info *);

float remove_last_cp(struct calibration_component_info *);

int find_nearest_test_point(float, struct test_point_info *,int);

int get_new_cp(struct calibration_component_info *);

void control_points_to_table( void *, Boolean, float, struct calibration_component_info * );

void xy_to_color(double, double, struct f_color *);

void draw_plot(struct calibration_component_info *);
void draw_all_plots(struct calibration_component_info *);
void draw_curve( struct calibration_component_info *, struct control_point_info *, const RGBColor * );
void draw_curve_2( struct calibration_component_info *, struct control_point_info *, const RGBColor * );
```

```
void draw_test_point_markers( struct calibration_component_info *, const RGBColor * );
void draw_current_point_marker( struct calibration_component_info *, const RGBColor * );
void draw_parent_markers( struct calibration_component_info *, const RGBColor * );
void draw_calibrated_point_markers( struct calibration_component_info *, const RGBColor * );
void draw_black_level_comp_markers( struct calibration_component_info * );
void draw_white_balance_markers( struct calibration_component_info * );
void draw_gamma_markers( struct calibration_component_info * );

void initialize_scale(struct scale_info *);
void to_screen(struct coordinates *,struct scale_info *);
void to_value(struct coordinates *,struct scale_info *);
void rotate_pos_45(struct coordinates *);
int lower_bulge(struct control_point_info *);
```

```
#include "cal_math.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

//
void initialize_control_point_ptrs( struct control_point_info *control_points )
{
    control_points->creation_order = NULL;
    control_points->x = NULL;
    control_points->y = NULL;
    control_points->shifted_x = NULL;
    control_points->y_for_shifted_x = NULL;
    control_points->indep = NULL;
    control_points->dep = NULL;
    control_points->d2 = NULL;
    control_points->temp = NULL;
    control_points->parents = NULL;
}

//
/* get_control_point_memory gets the memory for all of the control point arrays */
int get_control_point_memory(struct control_point_info *my_control_points)
{
    initialize_control_point_ptrs( my_control_points );

    // Use "bail" to clean up memory and avoid any possible leaks.

    if( ( my_control_points->creation_order = (int*)calloc( 1, INT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->x = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->y = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->shifted_x = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->y_for_shifted_x = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->indep = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->dep = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->d2 = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->temp = (float*)calloc( 1, FLOAT_CONTROL_POINTS ) ) == 0 )
        goto bail;

    if( ( my_control_points->parents = (struct parent_info *)calloc( 1, PARENT_INFO_CONTROL_POINTS ) ) == 0 )
        goto bail;

    /* some initializations */
    reset_control_points ( my_control_points );
    calculate_smoothing ( my_control_points );
    my_control_points->black_level = 0.0;
    my_control_points->white_level = 1.0;

    return 0;

bail:

    dump_control_point_memory( my_control_points );
    return -1;
}

//
/* dump_control_point_memory frees up all of the control point arrays */
void dump_control_point_memory(struct control_point_info *my_control_points)
{
    if ( my_control_points->parents )
        free( my_control_points->parents );

    if ( my_control_points->temp )
        free( my_control_points->temp );

    if ( my_control_points->d2 )
        free( my_control_points->d2 );

    if ( my_control_points->dep )
```

```
        free( my_control_points->dep );

    if ( my_control_points->indep )
        free( my_control_points->indep );

    if ( my_control_points->y_for_shifted_x )
        free( my_control_points->y_for_shifted_x );

    if ( my_control_points->shifted_x )
        free( my_control_points->shifted_x );

    if ( my_control_points->y )
        free( my_control_points->y );

    if ( my_control_points->x )
        free( my_control_points->x );

    if ( my_control_points->creation_order )
        free( my_control_points->creation_order );

    initialize_control_point_ptrs( my_control_points );
}

//

int stream_out_control_point_info( char *dest_data, struct control_point_info *my_control_points )
{
    int err = noErr;

    int count = my_control_points->count;
    if( count < 0 || count > MAX_CONTROL_POINTS )
        return( paramErr );

    if( NULL == my_control_points || NULL == dest_data )
        return( paramErr );

    // Need to pay attention to byte ordering if we want to go cross platform

    // By knowing the size of the data and simply incrementing the offset,
    // we can safely pull in bad data without crashing since we're not parsing the data
    // like we would have to if it were in a tagged format.

    UInt32 offset = 0;

    *((int*)(dest_data + offset)) = my_control_points->count;
    offset += sizeof(int);

    *((float*)(dest_data + offset)) = my_control_points->black_level;
    offset += sizeof(float);

    *((float*)(dest_data + offset)) = my_control_points->white_level;
    offset += sizeof(float);

    if ( my_control_points->creation_order )
    {
        memcpy( dest_data + offset, (const char *)my_control_points->creation_order, count * sizeof(int) );
    }
    offset += count * sizeof(int);

    if ( my_control_points->x )
    {
        memcpy( dest_data + offset, (const char *)my_control_points->x, count * sizeof(float) );
    }
    offset += count * sizeof(float);

    if ( my_control_points->y )
    {
        memcpy( dest_data + offset, (const char *)my_control_points->y, count * sizeof(float) );
    }
    offset += count * sizeof(float);

    /*
    if ( my_control_points->shifted_x )
    {
        memcpy( dest_data + offset, (const char *)my_control_points->shifted_x, count * sizeof(float) );
    }
    offset += count * sizeof(float);

    if ( my_control_points->y_for_shifted_x )
    {
        memcpy( dest_data + offset, (const char *)my_control_points->y_for_shifted_x, count * sizeof(float) );
    }
    offset += count * sizeof(float);

    if ( my_control_points->indep )
    {
        memcpy( dest_data + offset, (const char *)my_control_points->indep, count * sizeof(float) );
    }
    offset += count * sizeof(float);
    */
}
```

```
if ( my_control_points->dep )
{
    memcpy( dest_data + offset, (const char *)my_control_points->dep, count * sizeof(float) );
}
offset += count * sizeof(float);

if ( my_control_points->d2 )
{
    memcpy( dest_data + offset, (const char *)my_control_points->d2, count * sizeof(float) );
}
offset += count * sizeof(float);

if ( my_control_points->temp )
{
    memcpy( dest_data + offset, (const char *)my_control_points->temp, count * sizeof(float) );
}
offset += count * sizeof(float);
*/
if ( my_control_points->parents )
{
    memcpy( dest_data + offset, (const char *)my_control_points->parents, count * sizeof(struct parent_info) );
}

return( err );
}

//
int stream_in_control_point_info( struct control_point_info *my_control_points, char *source_data )
{
    int err = noErr;

    if( NULL == my_control_points || NULL == source_data )
        return( paramErr );

    reset_control_points ( my_control_points );

    UInt32 offset = 0;

    my_control_points->count = *((int*)(source_data + offset));
    offset += sizeof(int);

    // We need to check and see if the number of control points is valid
    int count = my_control_points->count;
    if( count < 0 || count > MAX_CONTROL_POINTS )
    {
        reset_control_points ( my_control_points );
        return( paramErr );
    }

    my_control_points->black_level = *((float*)(source_data + offset));
    offset += sizeof(float);

    my_control_points->white_level = *((float*)(source_data + offset));
    offset += sizeof(float);

    // Need to pay attention to byte ordering if we want to go cross platform

    if ( my_control_points->creation_order )
    {
        memcpy( (char *)my_control_points->creation_order, source_data + offset, count * sizeof(int) );
    }
    offset += count * sizeof(int);

    if ( my_control_points->x )
    {
        memcpy( (char *)my_control_points->x, source_data + offset, count * sizeof(float) );
    }
    offset += count * sizeof(float);

    if ( my_control_points->y )
    {
        memcpy( (char *)my_control_points->y, source_data + offset, count * sizeof(float) );
    }
    offset += count * sizeof(float);
/*
    if ( my_control_points->shifted_x )
    {
        memcpy( (char *)my_control_points->shifted_x, source_data + offset, count * sizeof(float) );
    }
    offset += count * sizeof(float);

    if ( my_control_points->y_for_shifted_x )
    {
        memcpy( (char *)my_control_points->y_for_shifted_x, source_data + offset, count * sizeof(float) );
    }
}
```

```
offset += count * sizeof(float);
if ( my_control_points->indep )
{
    memcpy( (char *)my_control_points->indep, source_data + offset, count * sizeof(float) );
}
offset += count * sizeof(float);
if ( my_control_points->dep )
{
    memcpy( (char *)my_control_points->dep, source_data + offset, count * sizeof(float) );
}
offset += count * sizeof(float);
if ( my_control_points->d2 )
{
    memcpy( (char *)my_control_points->d2, source_data + offset, count * sizeof(float) );
}
offset += count * sizeof(float);
if ( my_control_points->temp )
{
    memcpy( (char *)my_control_points->temp, source_data + offset, count * sizeof(float) );
}
offset += count * sizeof(float);
*/
if ( my_control_points->parents )
{
    memcpy( (char *)my_control_points->parents, source_data + offset, count * sizeof(struct parent_info) );
}

return( err );
}

//
UInt32 get_control_point_info_size( int count )
{
    UInt32 data_size = 0;

    if( count < 0 || count > MAX_CONTROL_POINTS )
        return 0;

    data_size = sizeof(int) + // count
                sizeof(float) + // black level
                sizeof(float) + // white level
                count * sizeof(int) + // creation order
                count * sizeof(float) + // x
                count * sizeof(float) + // y
                count * sizeof(float) + // shifted_x
                count * sizeof(float) + // y_for_shifted_x
                count * sizeof(float) + // indep
                count * sizeof(float) + // dep
                count * sizeof(float) + // d2
                count * sizeof(float) + // temp
                count * sizeof(struct parent_info); // parents

    return( data_size );
}

//
void print_control_point_info( struct control_point_info *cp_info )
{
    int i;

    DEBUG_VAR_PRINT("num calibrated points = %d", cp_info->count);

    DEBUG_VAR_PRINT("black_level = %f", cp_info->black_level);
    DEBUG_VAR_PRINT("white_level = %f", cp_info->white_level);

    DEBUG_PRINT("creation order:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%d", *(cp_info->creation_order + i) );
    }

    DEBUG_PRINT("input points:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->x + i) );
    }

    DEBUG_PRINT("output points:");
    for( i=0; i<cp_info->count; i++ )
    {

```

```
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->y + i) );
    }

    DEBUG_PRINT("shifted input points:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->shifted_x + i) );
    }

    DEBUG_PRINT("shifted output points:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->y_for_shifted_x + i) );
    }

    DEBUG_PRINT("diagonal points:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->indep + i) );
    }

    DEBUG_PRINT("diagonal deviation:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->dep + i) );
    }

    DEBUG_PRINT("smoothing derivatives:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->d2 + i) );
    }

    DEBUG_PRINT("smoothing temps:");
    for( i=0; i<cp_info->count; i++ )
    {
        DEBUG_VAR_PRINT("%d = ", i );
        DEBUG_EXTRA_VAR_PRINT("%f", *(cp_info->temp + i) );
    }

// struct parent_info *parents;    /* parents that were used to calibrate this point, */
}

//
#pragma mark -
//

void initialize_slider(struct slider_info *this_slider)
{
    this_slider->this_control = NULL;

    if((this_slider->slider_rect.right - this_slider->slider_rect.left) > (this_slider->slider_rect.bottom -
this_slider->slider_rect.top))
    {
        /* horizontal */
        this_slider->this_control =
NewControl(this_slider->my_window, &(this_slider->slider_rect), "\p", false, 0, 0, (short)(this_slider->slider_rect.
right-this_slider->slider_rect.left-25), (short)(kControlSliderProc+kControlSliderLiveFeedback+
kControlSliderReverseDirection), (SInt32)this_slider->data_pointer);
    }
    else
    {
        /* vertical */
        this_slider->this_control =
NewControl(this_slider->my_window, &(this_slider->slider_rect), "\p", false, 0, 0, (this_slider->slider_rect.bottom-
this_slider->slider_rect.top-25), kControlSliderProc+kControlSliderLiveFeedback+kControlSliderReverseDirection, (
SInt32)this_slider->data_pointer);
    }

    this_slider->action_function = NewControlActionUPP((ControlActionProcPtr)this_slider->live_function);
    SetControlAction(this_slider->this_control, this_slider->action_function);

    initialize_slider_scale(this_slider);

    this_slider->current_value = 0.0;
}

//
```

```
void dispose_slider(struct slider_info *this_slider)
{
    if ( this_slider->action_function )
        DisposeRoutineDescriptor( this_slider->action_function );
        DisposeControlActionUPP( this_slider->action_function );

    if ( this_slider->this_control )
        DisposeControl (this_slider->this_control );
}

//

void initialize_slider_scale(struct slider_info *this_slider)
{
    if((this_slider->slider_rect.right - this_slider->slider_rect.left) > (this_slider->slider_rect.bottom -
this_slider->slider_rect.top))
    {
        /* horizontal */
        this_slider->to_screen_scale =
(this_slider->slider_rect.right-this_slider->slider_rect.left-25)/(this_slider->slider_max -
this_slider->slider_min);
        this_slider->to_screen_offset = 0.5 + (this_slider->slider_rect.left + 12) - this_slider->to_screen_scale *
this_slider->slider_min;

        this_slider->to_value_scale = 1.0/this_slider->to_screen_scale;
        this_slider->to_value_offset = this_slider->slider_min;

        this_slider->to_position_scale = 1.0/this_slider->to_value_scale;
        this_slider->to_position_offset = 0.5 + this_slider->to_value_offset*this_slider->to_position_scale;
    }
    else
    {
        /* vertical */
        this_slider->to_screen_scale =
-(this_slider->slider_rect.bottom-this_slider->slider_rect.top-25)/(this_slider->slider_max -
this_slider->slider_min);
        this_slider->to_screen_offset = 0.5 + (this_slider->slider_rect.bottom - 13) - this_slider->to_screen_scale *
* this_slider->slider_min;

        this_slider->to_value_scale = -1.0/this_slider->to_screen_scale;
        this_slider->to_value_offset = this_slider->slider_min;

        this_slider->to_position_scale = 1.0/this_slider->to_value_scale;
        this_slider->to_position_offset = 0.5 - this_slider->to_value_offset*this_slider->to_position_scale;
    }
}

//

float slider_to_value(short position,struct slider_info *this_slider)
{
    this_slider->current_value = (float)position * this_slider->to_value_scale + this_slider->to_value_offset;
    return this_slider->current_value;
}

//

short value_to_screen(float value,struct slider_info *this_slider)
{
    return (short)(value * this_slider->to_screen_scale + this_slider->to_screen_offset);
}

//

void force_slider_position(float value,struct slider_info *this_slider)
{
    this_slider->current_value = value;

    SetControlValue(this_slider->this_control,(short)(value * this_slider->to_position_scale +
this_slider->to_position_offset));
}

//

#pragma mark -

//

void initialize_palette(WindowPtr my_window, PaletteHandle *my_palette)
{
    int i;
    RGBColor this_color;

    /* load in a grey ramp */

    *my_palette = NewPalette( 256, 0, pmExplicit + pmTolerant, 0 );
}
```



```
SetPalette(my_window,*my_palette,0);

for(i=0;i<256;i++)
{
    this_color.red = (unsigned short)i << 8;
    this_color.green = (unsigned short)i << 8;
    this_color.blue = (unsigned short)i << 8;

    SetEntryColor(*my_palette,i,&this_color);
}

ActivatePalette(my_window);
}

//
void value_to_comp_color(float value, struct f_color *f_color, struct calibration_component_info *this_component)
{
    f_color->red = value * this_component->component_color.red;
    f_color->green = value * this_component->component_color.green;
    f_color->blue = value * this_component->component_color.blue;
}

//
void offset_scale_f_pixel(struct f_color *f_color, struct calibration_component_info *this_component)
{
    f_color->red = f_color->red * (1.0 - this_component->cp_r->black_level) + this_component->cp_r->black_level;
    f_color->green = f_color->green * (1.0 - this_component->cp_g->black_level) +
this_component->cp_g->black_level;
    f_color->blue = f_color->blue * (1.0 - this_component->cp_b->black_level) + this_component->cp_b->black_level;
}

//
void color_float_to_pixel(struct f_color *f_color, struct components *pixel,struct calibration_component_info
*this_component)
{
    pixel->red = 0.5 + this_component->max_pixel_value * f_color->red;
    pixel->green = 0.5 + this_component->max_pixel_value * f_color->green;
    pixel->blue = 0.5 + this_component->max_pixel_value * f_color->blue;
}

//
#if TARGET_API_MAC_CARBON

int update_colors( struct graphics_dev_info *this_dev_info, struct components *colors, int count )
{
    if( ( NULL == this_dev_info ) || ( NULL == colors ) || ( count < 1 ) )
        return( paramErr );

    int err = noErr;

    // make sure it is a valid device
    if( ( NULL != this_dev_info->display_id ) && ( NULL != this_dev_info->gamma_table_w_header ) )
    {
        int index = 0;

        for( int i=0; i<count; i++ )
        {
            // We put the three gray patches for the patterns at indices 8, 16 and 24.
            index = 8 * (i+1);
            this_dev_info->gamma_table_w_header->u.table.data[ 0 + index ] = colors[i].red; // red channel
            this_dev_info->gamma_table_w_header->u.table.data[ 256 + index ] = colors[i].green; // green channel
            this_dev_info->gamma_table_w_header->u.table.data[ 512 + index ] = colors[i].blue; // blue channel
        }

        err = copy_gamma_to_dev( this_dev_info );
    }
    else
    {
        err = paramErr;
    }

    return( err );
}

#else

// This method works for 8-bit mode
int update_colors( struct graphics_dev_info *this_dev_info, struct components *colors, int count )
{
    if( ( NULL == this_dev_info ) || ( NULL == colors ) || ( count < 1 ) )
        return( paramErr );

    int err = noErr;
```

```
if( NULL != this_dev_info->display_id )
{
    GDHandle device = NULL;

    err = DMGetGDeviceByDisplayID ( this_dev_info->display_id, &device, false );
    if( noErr == err )
    {
        ColorSpec my_specs[1];
        VDSetEntryRecord vDSetEntryRecord;
        CntrlParam control;

        for(int i = 0; i < count; i++ )
        {
            my_specs[0].value = 0;
            my_specs[0].rgb.red   = (unsigned short)(colors[i].red ) << 8;
            my_specs[0].rgb.green = (unsigned short)(colors[i].green) << 8;
            my_specs[0].rgb.blue  = (unsigned short)(colors[i].blue ) << 8;

            vDSetEntryRecord.csTable = my_specs;
            vDSetEntryRecord.csStart = 8*(i+1);
            vDSetEntryRecord.csCount = 0;

            control.ioCompletion = NULL;
            control.ioCRefNum = (**device).gdRefNum;

            if( (**(**device).gdPMap).pixelType == 0 )
                control.csCode = cscSetEntries;
            else
                control.csCode = cscDirectSetEntries;

            *((Ptr *)&control.csParam[0]) = (Ptr)&vDSetEntryRecord;

            err = PBControlSync((ParmBlkPtr) &control);
        }
    }
    else
    {
        err = paramErr;
    }

    return( err );
}

#endif

//
/* reset_control_points sets control structure to basic bottom and top values */
void reset_control_points( struct control_point_info *control_points )
{
    control_points->creation_order[0] = 0;
    control_points->x[0] = 0.0;
    control_points->y[0] = 0.0;
    control_points->parents[0].a_x = 0.0;
    control_points->parents[0].a_y = 0.0;
    control_points->parents[0].b_x = 0.0;
    control_points->parents[0].b_y = 0.0;

    control_points->creation_order[1] = 1;
    control_points->x[1] = 1.0;
    control_points->y[1] = 1.0;
    control_points->parents[1].a_x = 1.0;
    control_points->parents[1].a_y = 1.0;
    control_points->parents[1].b_x = 1.0;
    control_points->parents[1].b_y = 1.0;

    control_points->count = 2;
}

//
/* calculate_smoothing figures the derivatives necessary to interpolate the curve along the diagonal */
void calculate_smoothing(struct control_point_info *my_control_points)
{
    int i,k;
    float p,qn,sig,un;

    /* rotate the data down 45 degrees */
    for(i=0;i<my_control_points->count;i++)
    {
        my_control_points->indep[i] = 0.707106781187 * (my_control_points->x[i] + my_control_points->y[i]);
        my_control_points->dep[i] = 0.707106781187 * (my_control_points->y[i] - my_control_points->x[i]);
    }

    /* now calculate the smoothing slopes */
    (my_control_points->d2)[0] = (my_control_points->temp)[0] = 0.0;
}
```

```
for (i=1; i < (my_control_points->count)-1; i++)
{
    sig = ( (my_control_points->indep)[i] - (my_control_points->indep)[i-1] ) /
        ( (my_control_points->indep)[i+1] - (my_control_points->indep)[i-1] );

    p = sig * (my_control_points->d2)[i-1] + 2.0;

    (my_control_points->d2)[i] = ( sig - 1.0 ) / p;

    (my_control_points->temp)[i] = ( (my_control_points->dep)[i+1] - (my_control_points->dep)[i] ) /
        ( (my_control_points->indep)[i+1] - (my_control_points->indep)[i] ) -
        ( (my_control_points->dep)[i] - (my_control_points->dep)[i-1] ) /
        ( (my_control_points->indep)[i] - (my_control_points->indep)[i-1] );

    (my_control_points->temp)[i] = ( 6.0 * (my_control_points->temp)[i] / ( (my_control_points->indep)[i+1] -
        (my_control_points->indep)[i-1] ) - sig * (my_control_points->temp)[i-1] ) / p;
}

qn = un = 0.0;

(my_control_points->d2)[(my_control_points->count)-1] = ( un - qn *
(my_control_points->temp)[(my_control_points->count)-2] ) /
    ( qn*(my_control_points->d2)[(my_control_points->count)-2] + 1.0);

for (k=(my_control_points->count)-2; k >= 0; k--)
{
    (my_control_points->d2)[k] = (my_control_points->d2)[k] * (my_control_points->d2)[k+1] +
(my_control_points->temp)[k];
}

//
void make_shifted_control_points(struct calibration_component_info *this_component)
{
    int i;

    for(i=0; i<this_component->cp_cur->count; i++)
    {
        this_component->cp_cur->shifted_x[i] = quantize_tab_position(this_component->cp_cur->x[i], this_component)
        this_component->cp_cur->y_for_shifted_x[i] =
get_y_from_x(this_component->cp_cur->shifted_x[i], this_component->cp_cur);
    }
}

//
/* locate_new_test_points builds up all the possible test points from the calibrated control points */
void locate_new_test_points(struct calibration_component_info *this_component)
{
    int i, j;
    float last_x;
    int new_count;

    /* reset the test point count back to zero */
    this_component->test_point_count = 0;

    /* iterate through the control points */
    for(i=0; i<this_component->cp_cur->count; i++)
    {
        /* each of the control points results in an inactive pre-calibrated test point */
        /* since this cannot be re-calibrated, the non-shifted real control points are used */
        this_component->test_points[this_component->test_point_count].x = this_component->cp_cur->x[i];
        this_component->test_points[this_component->test_point_count].y = this_component->cp_cur->y[i];
        this_component->test_points[this_component->test_point_count].parents = this_component->cp_cur->parents[i];
        this_component->test_points[this_component->test_point_count].point_status = 1;
        (this_component->test_point_count)++;

        /* the rest of the points are not calibrated */
        for(j=i+1; j<this_component->cp_cur->count; j++)
        {
            /* note that shifted, representable control points are used instead of the fractional actual control
points */
            this_component->test_points[this_component->test_point_count].x = (this_component->cp_cur->shifted_x[i]
this_component->cp_cur->shifted_x[j]) * 0.5;

            /* estimate the value for this test from the curve */
            this_component->test_points[this_component->test_point_count].y =
get_y_from_x(this_component->test_points[this_component->test_point_count].x, this_component->cp_cur);

            this_component->test_points[this_component->test_point_count].parents.a_x =
this_component->cp_cur->shifted_x[i];
            this_component->test_points[this_component->test_point_count].parents.a_y =
this_component->cp_cur->y_for_shifted_x[i];
            this_component->test_points[this_component->test_point_count].parents.b_x =
this_component->cp_cur->shifted_x[j];
            this_component->test_points[this_component->test_point_count].parents.b_y =
```

```
this_component->cp_cur->y_for_shifted_x[j];
    this_component->test_points[this_component->test_point_count].point_status = 0;
    (this_component->test_point_count)++;
}

/* sort the new list to get them in order and mark duplicates */
qsort((void *) (this_component->test_points), this_component->test_point_count, sizeof(struct
test_point_info), compare_points);

/* remove duplicates */
i = 0;
new_count = 0;
last_x = -1.0;

while(i < this_component->test_point_count)
{
    // if( this_component->test_points[i].x == last_x )
    if( ( this_component->test_points[i].x - last_x ) < ( 1.0 / 64.0 ) )
    {
        i++;
    }
    else
    {
        last_x = this_component->test_points[i].x;
        this_component->test_points[new_count++] = this_component->test_points[i++];
    }
}

this_component->test_point_count = new_count;
}

//
/* locate_value does some painful math to locate a function minimum */

float locate_value(float (*func)(float,float,struct control_point_info *), float x1, float x2, float tol,float
target,struct control_point_info *my_control_points)
{
    int iter;
    float a,b,c,d,e,min1,min2;
    float fa,fb,fc,p,q,r,s,toll,xm;

    a=x1;
    b=x2;
    c=x2;

    fa=(*func)(a,target,my_control_points);
    fb=(*func)(b,target,my_control_points);

    if ((fa > 0.0 && fb > 0.0) || (fa < 0.0 && fb < 0.0))
        return 0.0;

    fc=fb;

    for (iter=1;iter<=ITMAX;iter++)
    {
        if((fb > 0.0 && fc > 0.0) || (fb < 0.0 && fc < 0.0))
        {
            c=a;
            fc=fa;
            e=d=b-a;
        }

        if(fabs(fc) < fabs(fb))
        {
            a=b;
            b=c;
            c=a;
            fa=fb;
            fb=fc;
            fc=fa;
        }

        toll=2.0*EPS*fabs(b)+0.5*tol;

        xm=0.5*(c-b);

        if(fabs(xm) <= toll || fb == 0.0)
            return b;

        if(fabs(e) >= toll && fabs(fa) > fabs(fb))
        {
            s=fb/fa;

            if (a == c)
            {
                p=2.0*xm*s;
```

```
        q=1.0-s;
    }
    else
    {
        q=fa/fc;
        r=fb/fc;
        p=s*(2.0*xm*q*(q-r)-(b-a)*(r-1.0));
        q=(q-1.0)*(r-1.0)*(s-1.0);
    }

    if(p > 0.0)
        q = -q;

    p=fabs(p);

    min1=3.0*xm*q-fabs(toll*q);

    min2=fabs(e*q);

    if(2.0*p < (min1 < min2 ? min1 : min2))
    {
        e=d;
        d=p/q;
    }
    else
    {
        d=xm;
        e=d;
    }
}
else
{
    d=xm;
    e=d;
}

a=b;

fa=fb;

if(fabs(d) > toll)
    b += d;
else
    b += SIGN(toll,xm);

fb = (*func)(b,target,my_control_points);
}

return 0.0;
}

//
/* compare_points is used by the sorting routine to generate a kinda complex ranking */
int compare_points(const void *first,const void *second)
{
    float f_diff;

    /* first check to see if it is a duplicate and force to bottom */
    f_diff = ((struct test_point_info *)first)->x - ((struct test_point_info *)second)->x;

    if( f_diff == 0.0 )
    {
        /* they are the same so give priority to a calibrated one */
        if(((struct test_point_info *)first)->point_status == 1)
        {
            /* first one calibrated, give it priority */
            return -1;
        }
        else
        {
            if(((struct test_point_info *)second)->point_status == 1)
            {
                /* only second one calibrated , give it priority */
                return 1;
            }
            else
            {
                /* neither calibrated, sort on spread of parents */
                f_diff = (((struct test_point_info *)first)->parents.b_x - ((struct test_point_info *)first)->parents.a_x) - (((struct test_point_info *)second)->parents.b_x - ((struct test_point_info *)second)->parents.a_x);

                if(f_diff >= 0.0)
                    return 1;
                else
                    return -1;
            }
        }
    }
}
```

```
    }
  }
  else if(f_diff > 0.0)
    return 1;
  else
    return -1;
}

//
/* set_adj_scale scales the adjustment slider between the nearest existing control points */
void set_adj_scale(struct calibration_component_info *this_component)
{
  int klo,khi,k;

  /* bracket this test point in the control point list */
  klo = 0;
  khi = this_component->cp_cur->count;

  while (khi-klo > 1)
  {
    k=(khi+klo) >> 1;

    if (this_component->cp_cur->x[k] > this_component->test_points[this_component->nearest_point].x)
      khi = k;
    else
      klo = k;
  }

  this_component->adj_slider->slider_min = this_component->cp_cur->y[khi-1];
  this_component->adj_slider->slider_max = this_component->cp_cur->y[khi];

  initialize_slider_scale(this_component->adj_slider);

  force_slider_position(get_y_from_x(this_component->test_points[this_component->nearest_point].x,this_component->
  cp_cur),this_component->adj_slider);
}

//
float quantitize_tab_position(float input,struct calibration_component_info *this_component)
{
  return (float)((int)(input * (float)(this_component->gam_tab_count - 1) +
  0.5))/((float)(this_component->gam_tab_count - 1));
}

//
/* get_y_from_x locates y along the curve from x */
float get_y_from_x(float x,void *my_control_points)
{
  float param,y;

  /* keep the input within limits */
  if(x >= 1.0)
    return 1.0;
  if(x <= 0.0)
    return 0.0;

  /* iterate to a solution */
  param = locate_value(param_from_x,0.0,1.41421356237,0.001,x,(struct control_point_info *)my_control_points);

  /* do the diagonal conversion */
  y = 0.707106781187 * (param + interpolate_value(param,(struct control_point_info *)my_control_points));

  /* keep the output within limits */
  if(y >= 1.0)
    return 1.0;
  if(y <= 0.0)
    return 0.0;

  return y;
}

//
/* param_from_x necessary function for get_y_from_x iteration */
float param_from_x(float param,float target_x,struct control_point_info *my_control_points)
{
  return ( (0.707106781187 * (param - interpolate_value(param,my_control_points))) - target_x );
}

//
/* percept_to_lum maps 0-1 perceptual changes to 0-1 luminance */
```

```
float percept_to_lum(float percept)
{
    if(percept > 0.08)
        return
(0.002624133830825372+percept*(0.04920250932797572+percept*(0.3075156832998482+percept*0.6406576735413507)));
    else
        return (percept * 0.1107419712070875);
}

//
/* lum_to_percept returns a perceptual value from a luminance */
float lum_to_percept(float lum)
{
    if(lum > 0.008856)
        return (1.16 * pow(lum,float(1.0/3.0)) - 0.16);
    else
        return lum * 9.03;
}

//
/* interpolate_value finds a value along the curve, but still on the diagonal */
float interpolate_value(float indep,struct control_point_info *my_control_points)
{
    int klo,khi,k;
    float h,b,a;
    double a2,b2;
    float dep;

    /* bracket position in control points */
    klo = 0;
    khi = my_control_points->count - 1;

    while (khi-klo > 1)
    {
        k=(khi+klo) >> 1;

        if (my_control_points->indep[k] > indep)
            khi = k;
        else
            klo = k;
    }

    h = my_control_points->indep[khi] - my_control_points->indep[klo];

    if (h == 0.0)
    {
        fprintf(stderr,"Bad xa input to routine get_background_pixel\n");
        exit(-1);
    }

    /* do the common math */
    a = ( my_control_points->indep[khi] - indep ) / h;
    b = ( indep - my_control_points->indep[klo] ) / h;

    a2 = a*a*a-a;
    b2 = b*b*b-b;

    /* with these two lines, we'd be using straight line interpolation instead of cubic */
    /* a2 = 0.0; */
    /* b2 = 0.0; */

    h = (h*h)/6.0;

    /* calculate each one and clip */
    dep = a * my_control_points->dep[klo] +
        b * my_control_points->dep[khi] +
        (a2 * my_control_points->d2[klo] +
        b2 * my_control_points->d2[khi]) * h;

    return dep;
}

//
/* remove_last_cp eliminates the last done control point */
float remove_last_cp(struct calibration_component_info *this_component)
{
    int k;
    float removed_x;

    k = 0;

    /* find the last one you made */

```

```
while(this_component->cp_cur->creation_order[k] != (this_component->cp_cur->count - 1) )
    k++;

removed_x = this_component->cp_cur->x[k];

/* go up one */
k++;

/* now skootch all the control points down one to fill in the space */
for(;k<this_component->cp_cur->count;k++)
{
    this_component->cp_cur->creation_order[k-1] = this_component->cp_cur->creation_order[k];
    this_component->cp_cur->x[k-1] = this_component->cp_cur->x[k];
    this_component->cp_cur->y[k-1] = this_component->cp_cur->y[k];
    this_component->cp_cur->parents[k-1] = this_component->cp_cur->parents[k];
}

/* there's one less point now */
this_component->cp_cur->count--;

return removed_x;
}

//
/* find_nearest_test_point locates the closest test point to a certain x */
int find_nearest_test_point(float fx, struct test_point_info *test_points,int test_point_count)
{
    int klo,khi,k;

    /* force in range */
    if(fx < 0.0 )
        return 0;
    if(fx >= 1.0 )
        return test_point_count-1;

    /* bracket position in test points */
    klo = 0;
    khi = test_point_count;

    while (khi-klo > 1)
    {
        k=(khi+klo) >> 1;

        if (test_points[k].x > fx)
            khi = k;
        else
            klo = k;
    }

    if( (test_points[khi].x - fx) < (fx - test_points[klo].x) )
        return khi;
    else
        return klo;
}

//
/* get_new_cp move control points around and get index to opening */
int get_new_cp(struct calibration_component_info *this_component)
{
    int klo,khi,k;

    /* bracket this test point in the control point list */
    klo = 0;
    khi = this_component->cp_cur->count;

    while (khi-klo > 1)
    {
        k=(khi+klo) >> 1;

        if (this_component->cp_cur->x[k] > this_component->test_points[this_component->nearest_point].x)
            khi = k;
        else
            klo = k;
    }

    /* skootch the control points up to make room for new point */
    for(k=this_component->cp_cur->count;k>khi;k--)
    {
        this_component->cp_cur->creation_order[k] = this_component->cp_cur->creation_order[k-1];
        this_component->cp_cur->x[k] = this_component->cp_cur->x[k-1];
        this_component->cp_cur->y[k] = this_component->cp_cur->y[k-1];
        this_component->cp_cur->parents[k] = this_component->cp_cur->parents[k-1];
    }
}
```



```
this_component->cp_cur->creation_order[khi] = this_component->cp_cur->count;
this_component->cp_cur->x[khi] = this_component->test_points[this_component->nearest_point].x;
this_component->cp_cur->parents[khi] = this_component->test_points[this_component->nearest_point].parents;

this_component->cp_cur->count++;

return khi;
}

//
void control_points_to_table( void *vc_gamma, Boolean target_perceptual, float target_gamma, struct
calibration_component_info *this_component )
{
    DEBUG_PRINT("Entered control_points_to_table()");

    UInt16 channel_count;
    if( this_component->this_dev_info->channel_count > 1 )
        channel_count = 3;
    else
        channel_count = 1;

    // CLip the gamma
    if( target_gamma < 1.0 )
        target_gamma = 1.0;
    else if( target_gamma > 3.0 )
        target_gamma = 3.0;

    UInt16 entry_count = this_component->this_dev_info->entry_count;
    UInt16 entry_size = this_component->this_dev_info->entry_size;
    UInt16 entry_size_bits = entry_size * 8;
    float max = 0.5 + this_component->this_dev_info->max_value;

#ifdef TARGET_API_MAC_CARBON
    ((CMVideoCardGamma *)vc_gamma)->tagType = cmVideoCardGammaTableType;
    ((CMVideoCardGamma *)vc_gamma)->u.table.channels = channel_count;
    ((CMVideoCardGamma *)vc_gamma)->u.table.entryCount = entry_count;
    ((CMVideoCardGamma *)vc_gamma)->u.table.entrySize = entry_size;

    char *data = ((CMVideoCardGamma *)vc_gamma)->u.table.data;
#else
    ((GammaTbl *)vc_gamma)->gVersion = 0;
    ((GammaTbl *)vc_gamma)->gType = 0;
    ((GammaTbl *)vc_gamma)->gFormulaSize = 0;
    ((GammaTbl *)vc_gamma)->gChanCnt = channel_count;
    ((GammaTbl *)vc_gamma)->gDataCnt = entry_count;
    ((GammaTbl *)vc_gamma)->gDataWidth = entry_size_bits;

    SInt16 *data = ((GammaTbl *)vc_gamma)->gFormulaData;
#endif

    UInt32 r_offset = entry_count * 0;
    UInt32 g_offset = entry_count * 1;
    UInt32 b_offset = entry_count * 2;

    float f_x;

    if( entry_size == 1 ) // 1 byte entries
    {
        // We fix the zero index so that we don't mess up video cards with hardware cursors.
        ((UInt8 *)data)[0 + r_offset] = (UInt8)0;

        if( channel_count == 3 )
        {
            ((UInt8 *)data)[0 + g_offset] = (UInt8)0;
            ((UInt8 *)data)[0 + b_offset] = (UInt8)0;
        }

        for( int i = 1; i < entry_count; i++ )
        {
            f_x = (float)i/(float)(entry_count - 1);

            if( target_perceptual == true )
            {
                ((UInt8 *)data)[i + r_offset] = (UInt8)( max * FULL_X_TO_Y_PTR ( percept_to_lum(f_x),
this_component->cp_r ) );

                if( channel_count == 3 )
                {
                    ((UInt8 *)data)[i + g_offset] = (UInt8)( max * FULL_X_TO_Y_PTR ( percept_to_lum(f_x),
this_component->cp_g ) );
                    ((UInt8 *)data)[i + b_offset] = (UInt8)( max * FULL_X_TO_Y_PTR ( percept_to_lum(f_x),
this_component->cp_b ) );
                }
            }
        }
    }
}
```

```
    }
    else
    {
        ((UInt8 *)data)[i + r_offset] = (UInt8)( max * FULL_X_TO_Y_PTR ( exp ( log(f_x) * target_gamma ),
this_component->cp_r ) );
        if( channel_count == 3 )
        {
            ((UInt8 *)data)[i + g_offset] = (UInt8)( max * FULL_X_TO_Y_PTR ( exp ( log(f_x) * target_gamma
this_component->cp_g ) );
            ((UInt8 *)data)[i + b_offset] = (UInt8)( max * FULL_X_TO_Y_PTR ( exp ( log(f_x) * target_gamma
this_component->cp_b ) );
        }
    }
}
else if( entry_size == 2 ) // 2 byte entries
{
    // We fix the zero index so that we don't mess up video cards with hardware cursors.
    ((UInt16 *)data)[0 + r_offset] = (UInt16)0;
    if( channel_count == 3 )
    {
        ((UInt16 *)data)[0 + g_offset] = (UInt16)0;
        ((UInt16 *)data)[0 + b_offset] = (UInt16)0;
    }
    for ( int i = 1; i < entry_count; i++ )
    {
        f_x = (float)i/(float)(entry_count - 1);
        if( target_perceptual == true )
        {
            ((UInt16 *)data)[i + r_offset] = (UInt16)( max * FULL_X_TO_Y_PTR ( percept_to_lum(f_x),
this_component->cp_r ) );
            if( channel_count == 3 )
            {
                ((UInt16 *)data)[i + g_offset] = (UInt16)( max * FULL_X_TO_Y_PTR ( percept_to_lum(f_x),
this_component->cp_g ) );
                ((UInt16 *)data)[i + b_offset] = (UInt16)( max * FULL_X_TO_Y_PTR ( percept_to_lum(f_x),
this_component->cp_b ) );
            }
        }
        else
        {
            ((UInt16 *)data)[i + r_offset] = (UInt16)( max * FULL_X_TO_Y_PTR ( exp ( log(f_x) * target_gamma ),
this_component->cp_r ) );
            if( channel_count == 3 )
            {
                ((UInt16 *)data)[i + g_offset] = (UInt16)( max * FULL_X_TO_Y_PTR ( exp ( log(f_x) * target_gamm
), this_component->cp_g ) );
                ((UInt16 *)data)[i + b_offset] = (UInt16)( max * FULL_X_TO_Y_PTR ( exp ( log(f_x) * target_gamm
), this_component->cp_b ) );
            }
        }
    }
}
else
{
    // not handled
}
DEBUG_PRINT("Left control_points_to_table()");
}

// _____
#pragma mark -
// _____

void xy_to_color(double x,double y,struct f_color *return_color)
{
    double scaler,x_sq,y_sq,y_cu,x_y;
    float biggest;
    x_sq = x*x;
    y_sq = y*y;
    y_cu = y_sq*y;
    x_y = x * y;
    scaler = 1.0/((12.17665516649234 + y)*(-28.82730863279437*x + 12.17665516649234*y -
2.367424242424242*x_y + y_sq));
```

```
return_color->red = (float)(0.6314851168815854*(-356.0680258743614*x - 212.0072213741749*x_sq +
150.4031341293302*y + 31.48268329945965*x_y -
17.41095715328913*x_sq*y + 24.52841614459813*y_sq +
4.986964059125081*x*y_sq + y_cu)*scaler);

return_color->green = (float)(1.068739459259297*(-210.4295418165587*x + 38.22400207126936*x_sq +
88.8854384633144*y - 62.25451833865149*x_y +
3.139121667537565*x_sq*y + 19.47631482244687*y_sq -
3.693389234792109*x*y_sq + y_cu)*scaler);

return_color->blue = (float)(3.571724767697527*(-62.96090614747237*x + 3.356985359261552*x_sq +
26.59468675669233*y + 22.23869395923638*x_y +
0.2756902707156631*x_sq*y - 9.99258356447512*y_sq +
2.250972672073948*x*y_sq - y_cu)*scaler);

biggest = return_color->red;

if(return_color->green > biggest)
    biggest = return_color->green;

if(return_color->blue > biggest)
    biggest = return_color->blue;

// biggest = 1.0;

return_color->red /= biggest;
return_color->green /= biggest;
return_color->blue /= biggest;
}

//

#pragma mark -

//

void draw_plot(struct calibration_component_info *this_component)
{
    Rect frame_rect;
    Rect bounds_rect;
    struct coordinates these_coordinates;
    char temp_text[64];

    // Save the clip region
    RgnHandle saved_clip_rgn = NewRgn();
    GetClip ( saved_clip_rgn );

    // Convert the plot coordinates to screen coordinates
    // for the top left of the plot frame...
    these_coordinates.fx = 0.0;
    these_coordinates.fy = 0.0;
    to_screen(&these_coordinates,this_component->plot_scale);
    frame_rect.left = these_coordinates.sx;
    frame_rect.bottom = these_coordinates.sy + 1;

    // ...and the bottom right of the plot frame.
    these_coordinates.fx = 1.0;
    these_coordinates.fy = 1.0;
    to_screen(&these_coordinates,this_component->plot_scale);
    frame_rect.right = these_coordinates.sx + 1;
    frame_rect.top = these_coordinates.sy;

    // Erase the rect behind the plot, including a little
    // extra on top and bottom for warning labels.
    bounds_rect = frame_rect;
    InsetRect(&bounds_rect,-5,-10);
    ClipRect(&bounds_rect);
    RGBBackColor(&RGB_GRAY_000);
    EraseRect(&bounds_rect);

    // If the curve bulges outside the plot frame, we display a warning.
    if(lower_bulge(this_component->cp_cur))
    {
        RGBForeColor(&RGB_RED);

        these_coordinates.fx = 0.0;
        these_coordinates.fy = 1.01;
        to_screen(&these_coordinates,this_component->plot_scale);

        MoveTo(these_coordinates.sx,these_coordinates.sy);
        sprintf(temp_text,"WARNING");
        DrawText(temp_text,0,strlen(temp_text));
    }

    // Point that is the current point
    // i == this_component->nearest_point
```

```
// Point has been calibrated
// test_points[i].point_status == 1

// Draw vertical lines for all the test points
draw_test_point_markers( this_component, &RGB_GRAY_064 );

// Now indicate the actual line that we're on.
draw_current_point_marker( this_component, &RGB_GRAY_255 );

// Draw pointers that indicate the parents.
draw_parent_markers( this_component, &RGB_GRAY_119 );

RGBColor which_color;

if( this_component->cp_cur == this_component->cp_r )
    which_color = RGB_RED;
else if( this_component->cp_cur == this_component->cp_g )
    which_color = RGB_GREEN;
else // this_component->cp_b
    which_color = RGB_BLUE;

ClipRect(&frame_rect);
draw_curve( this_component, this_component->cp_cur, &which_color );
ClipRect(&bounds_rect);

// Draw markers on the calibrated points
draw_calibrated_point_markers( this_component, &RGB_GRAY_170 );

// Draw the frame
RGBForeColor(&RGB_GRAY_255);
FrameRect(&frame_rect);

// Restore the old clip region
SetClip( saved_clip_rgn );
DisposeRgn( saved_clip_rgn );
}

//
void draw_all_plots(struct calibration_component_info *this_component)
{
    Rect frame_rect;
    Rect bounds_rect;
    struct coordinates these_coordinates;
    char temp_text[64];

    // Save the clip region
    RgnHandle saved_clip_rgn = NewRgn();
    GetClip ( saved_clip_rgn );

    // Convert the plot coordinates to screen coordinates
    // for the top left of the plot frame...
    these_coordinates.fx = 0.0;
    these_coordinates.fy = 0.0;
    to_screen(&these_coordinates,this_component->plot_scale);
    frame_rect.left = these_coordinates.sx;
    frame_rect.bottom = these_coordinates.sy + 1;

    // ...and the bottom right of the plot frame.
    these_coordinates.fx = 1.0;
    these_coordinates.fy = 1.0;
    to_screen(&these_coordinates,this_component->plot_scale);
    frame_rect.right = these_coordinates.sx + 1;
    frame_rect.top = these_coordinates.sy;

    // Erase the area
    bounds_rect = frame_rect;
    InsetRect( &bounds_rect, -120, -10 );
    ClipRect(&bounds_rect);
    RGBBackColor(&RGB_GRAY_000);
    EraseRect(&bounds_rect);

    // If the curve bulges outside the plot frame, we display a warning.
    if(lower_bulge(this_component->cp_cur))
    {
        RGBForeColor(&RGB_RED);

        these_coordinates.fx = 0.0;
        these_coordinates.fy = 1.01;
        to_screen(&these_coordinates,this_component->plot_scale);

        MoveTo(these_coordinates.sx,these_coordinates.sy);
        sprintf(temp_text,"WARNING");
        DrawText(temp_text,0,strlen(temp_text));
    }

    // Point that is the current point
```

```
// i == this_component->nearest_point

// Point has been calibrated
// test_points[i].point_status == 1

// Draw vertical lines for all the test points
// draw_test_point_markers( this_component, &RGB_GRAY_064 );

// Now indicate the actual line that we're on.
// draw_current_point_marker( this_component, &RGB_GRAY_255 );

// Draw pointers that indicate the parents.
// draw_parent_markers( this_component, &RGB_GRAY_119 );

PenState pen_state;
GetPenState( &pen_state );
PenMode ( addOver );

ClipRect(&frame_rect);

draw_curve_2( this_component, this_component->cp_b, &RGB_BLUE );
this_component->cp_cur = this_component->cp_b;
// draw_calibrated_point_markers( this_component, &RGB_GRAY_170 );

draw_curve_2( this_component, this_component->cp_g, &RGB_GREEN );
this_component->cp_cur = this_component->cp_g;
// draw_calibrated_point_markers( this_component, &RGB_GRAY_170 );

draw_curve_2( this_component, this_component->cp_r, &RGB_RED );
this_component->cp_cur = this_component->cp_r;
// draw_calibrated_point_markers( this_component, &RGB_GRAY_170 );

ClipRect(&bounds_rect);

// Draw markers on the calibrated points
// draw_calibrated_point_markers( this_component, &RGB_GRAY_170 );

draw_black_level_comp_markers( this_component );
draw_white_balance_markers( this_component );
draw_gamma_markers( this_component );

SetPenState( &pen_state );

// Draw the frame
RGBForeColor(&RGB_GRAY_255);
FrameRect(&frame_rect);

// Restore the old clip region
SetClip( saved_clip_rgn );
DisposeRgn( saved_clip_rgn );
}

//

void draw_curve( struct calibration_component_info *this_component, struct control_point_info *which_cp, const
RGBColor *color )
{
    struct coordinates these_coordinates;

    // f_step = 1.41421356237/sqrt(this_component->plot_scale->to_screen_x_scale *
    this_component->plot_scale->to_screen_x_scale
    // + this_component->plot_scale->to_screen_y_scale *
    this_component->plot_scale->to_screen_y_scale);

    RGBForeColor(color);

    float f_step = 1 / this_component->plot_scale->to_screen_x_scale;
    these_coordinates.fx = 0.0;
    these_coordinates.fy = 0.0;
    to_screen(&these_coordinates, this_component->plot_scale);
    MoveTo(these_coordinates.sx, these_coordinates.sy);

    for( float f_x = 0.0; f_x <= 1.41421356237; f_x += f_step )
    {
        these_coordinates.fx = f_x;
        these_coordinates.fy = interpolate_value(these_coordinates.fx, which_cp);
        rotate_pos_45(&these_coordinates);
        to_screen(&these_coordinates, this_component->plot_scale);
        LineTo(these_coordinates.sx, these_coordinates.sy);
    }
}

//

void draw_curve_2( struct calibration_component_info *this_component, struct control_point_info *which_cp, const
RGBColor *color )
{
    struct coordinates these_coordinates;
```

```
//float target_gamma = 1.0;

RGBForeColor(color);

float f_step = 1 / this_component->plot_scale->to_screen_x_scale;
these_coordinates.fx = this_component->plot_scale->x_min;
these_coordinates.fy = this_component->plot_scale->y_min;
to_screen(&these_coordinates,this_component->plot_scale);
MoveTo(these_coordinates.sx,these_coordinates.sy);

if( this_component->plot_target_perceptual )
{
    for( float f_x = this_component->plot_scale->x_min; f_x <= this_component->plot_scale->x_max; f_x += f_step
    )
    {
        these_coordinates.fx = f_x;
        these_coordinates.fy = FULL_X_TO_Y_PTR ( percept_to_lum(f_x), which_cp );
        to_screen(&these_coordinates,this_component->plot_scale);
        LineTo(these_coordinates.sx,these_coordinates.sy);
    }
}
else
{
    for( float f_x = this_component->plot_scale->x_min; f_x <= this_component->plot_scale->x_max; f_x += f_step
    )
    {
        these_coordinates.fx = f_x;
        these_coordinates.fy = FULL_X_TO_Y_PTR ( exp ( log(f_x) * this_component->plot_target_gamma ), which_cp
    );
        to_screen(&these_coordinates,this_component->plot_scale);
        LineTo(these_coordinates.sx,these_coordinates.sy);
    }
}

//
void draw_test_point_markers( struct calibration_component_info *this_component, const RGBColor *color )
{
    struct coordinates these_coordinates;
    int i;

    RGBForeColor( color );

    // Draw vertical lines for all the test points
    for( i=0; i < this_component->test_point_count; i++ )
    {
        // If the point has not been calibrated, we draw a line to show that it's a possible calibration point.
        if( this_component->test_points[i].point_status != 1 )
        {
            these_coordinates.fx = this_component->test_points[i].x;
            these_coordinates.fy = 0.0;
            to_screen(&these_coordinates,this_component->plot_scale);

            MoveTo(these_coordinates.sx,these_coordinates.sy - 2);

            these_coordinates.fx = this_component->test_points[i].x;
            these_coordinates.fy = 1.0;
            to_screen(&these_coordinates,this_component->plot_scale);

            LineTo(these_coordinates.sx,these_coordinates.sy + 2);
        }
    }
}

//
void draw_current_point_marker( struct calibration_component_info *this_component, const RGBColor *color )
{
    struct coordinates these_coordinates;

    RGBForeColor( color );

    // Now indicate the actual line that we're on.
    these_coordinates.fx = this_component->test_points[this_component->nearest_point].x;
    these_coordinates.fy = 0.0;
    to_screen(&these_coordinates,this_component->plot_scale);

    MoveTo(these_coordinates.sx,these_coordinates.sy - 2);

    these_coordinates.fx = this_component->test_points[this_component->nearest_point].x;
    these_coordinates.fy = 1.0;
    to_screen(&these_coordinates,this_component->plot_scale);

    LineTo(these_coordinates.sx,these_coordinates.sy + 2);
}

//
```

```
void draw_parent_markers( struct calibration_component_info *this_component, const RGBColor *color )
{
    struct coordinates these_coordinates;
    short nearest_point_x;

    RGBForeColor( color );

    // Now indicate the actual line that we're on.
    these_coordinates.fx = this_component->test_points[this_component->nearest_point].x;
    these_coordinates.fy = 0.0;
    to_screen(&these_coordinates,this_component->plot_scale);

    nearest_point_x = these_coordinates.sx;

    // Parent A
    these_coordinates.fx = this_component->test_points[this_component->nearest_point].parents.a_x;
    these_coordinates.fy = 0.0;
    to_screen(&these_coordinates,this_component->plot_scale);

    MoveTo(these_coordinates.sx,these_coordinates.sy + 2);
    LineTo(these_coordinates.sx,these_coordinates.sy + 7);
    LineTo(nearest_point_x - 5,these_coordinates.sy + 7);
    LineTo(nearest_point_x - 1,these_coordinates.sy + 3);

    MoveTo(these_coordinates.sx - 2,these_coordinates.sy + 4);
    LineTo(these_coordinates.sx,these_coordinates.sy + 2);
    LineTo(these_coordinates.sx + 2,these_coordinates.sy + 4);

    // Parent B
    these_coordinates.fx = this_component->test_points[this_component->nearest_point].parents.b_x;
    these_coordinates.fy = 0.0;
    to_screen(&these_coordinates,this_component->plot_scale);

    MoveTo(these_coordinates.sx,these_coordinates.sy + 2);
    LineTo(these_coordinates.sx,these_coordinates.sy + 7);
    LineTo(nearest_point_x + 5,these_coordinates.sy + 7);
    LineTo(nearest_point_x + 1,these_coordinates.sy + 3);

    MoveTo(these_coordinates.sx - 2,these_coordinates.sy + 4);
    LineTo(these_coordinates.sx,these_coordinates.sy + 2);
    LineTo(these_coordinates.sx + 2,these_coordinates.sy + 4);
}

//
void draw_calibrated_point_markers( struct calibration_component_info *this_component, const RGBColor *color )
{
    struct coordinates these_coordinates;
    Rect temp_rect;
    int i;

    RGBForeColor( color );

    // Draw markers on the calibrated points
    for( i=0; i < this_component->test_point_count; i++ )
    {
        // Has it been calibrated?
        if( this_component->test_points[i].point_status == 1 )
        {
            // If it's an end, ignore it.
            if ( this_component->test_points[i].x == 0.0 || this_component->test_points[i].x == 1.0 )
                continue;

            these_coordinates.fx = this_component->test_points[i].x;
            these_coordinates.fy = this_component->test_points[i].y;
            to_screen(&these_coordinates,this_component->plot_scale);

            // If it's the current point and the user hasn't advanced yet, we put a dot on it.
            if( i == this_component->nearest_point && !this_component->new_point )
            {
                temp_rect.left = these_coordinates.sx - 2;
                temp_rect.top = these_coordinates.sy - 2;
                temp_rect.right = temp_rect.left + 5;
                temp_rect.bottom = temp_rect.top + 5;

                PaintOval(&temp_rect);
            }
            else // Otherwise, we put a lock on it.
            {
                temp_rect.left = these_coordinates.sx - 2;
                temp_rect.top = these_coordinates.sy - 1;
                temp_rect.right = temp_rect.left + 5;
                temp_rect.bottom = temp_rect.top + 4;

                PaintRect(&temp_rect);

                OffsetRect(&temp_rect,0,-3);
            }
        }
    }
}
```

```
        FrameOval(&temp_rect);
    }
}
}
}
//
void draw_black_level_comp_markers( struct calibration_component_info *this_component )
{
    struct coordinates red_coords, green_coords, blue_coords, black_coords;
    Point line_start;
    float target_gamma = 1.0;

    black_coords.fx = this_component->plot_scale->x_min;
    black_coords.fy = this_component->plot_scale->y_min;
    to_screen(&black_coords, this_component->plot_scale);

    red_coords.fx = this_component->plot_scale->x_min;
    red_coords.fy = FULL_X_TO_Y_PTR ( exp ( log(red_coords.fx) * target_gamma ), this_component->cp_r );
    to_screen(&red_coords, this_component->plot_scale);

    green_coords.fx = this_component->plot_scale->x_min;
    green_coords.fy = FULL_X_TO_Y_PTR ( exp ( log(green_coords.fx) * target_gamma ), this_component->cp_g );
    to_screen(&green_coords, this_component->plot_scale);

    blue_coords.fx = this_component->plot_scale->x_min;
    blue_coords.fy = FULL_X_TO_Y_PTR ( exp ( log(blue_coords.fx) * target_gamma ), this_component->cp_b );
    to_screen(&blue_coords, this_component->plot_scale);

    line_start.h = black_coords.sx - 20;
    line_start.v = ( red_coords.sy + green_coords.sy + blue_coords.sy ) / 3;

    RGBForeColor( &RGB_BLUE );
    MoveTo( blue_coords.sx, blue_coords.sy );
    LineTo( blue_coords.sx - 10, blue_coords.sy );
    LineTo( blue_coords.sx - 10, line_start.v );
    LineTo( blue_coords.sx - 20, line_start.v );

    MoveTo( blue_coords.sx - 3, blue_coords.sy - 2 );
    LineTo( blue_coords.sx - 1, blue_coords.sy );
    LineTo( blue_coords.sx - 3, blue_coords.sy + 2 );

    RGBForeColor( &RGB_GREEN );
    MoveTo( green_coords.sx, green_coords.sy );
    LineTo( green_coords.sx - 10, green_coords.sy );
    LineTo( green_coords.sx - 10, line_start.v );
    LineTo( green_coords.sx - 20, line_start.v );

    MoveTo( green_coords.sx - 3, green_coords.sy - 2 );
    LineTo( green_coords.sx - 1, green_coords.sy );
    LineTo( green_coords.sx - 3, green_coords.sy + 2 );

    RGBForeColor( &RGB_RED );
    MoveTo( red_coords.sx, red_coords.sy );
    LineTo( red_coords.sx - 10, red_coords.sy );
    LineTo( red_coords.sx - 10, line_start.v );
    LineTo( red_coords.sx - 20, line_start.v );

    MoveTo( red_coords.sx - 3, red_coords.sy - 2 );
    LineTo( red_coords.sx - 1, red_coords.sy );
    LineTo( red_coords.sx - 3, red_coords.sy + 2 );

    RGBForeColor( &RGB_WHITE );
    MoveTo( line_start.h - 80, line_start.v - 9 );
    DrawString("\pBlack Level");
    MoveTo( line_start.h - 80, line_start.v + 1 );
    DrawString("\pCompensation");
}
//
void draw_white_balance_markers( struct calibration_component_info *this_component )
{
    struct coordinates red_coords, green_coords, blue_coords, white_coords;
    Point line_start;
    float target_gamma = 1.0;

    white_coords.fx = this_component->plot_scale->x_max;
    white_coords.fy = this_component->plot_scale->y_max;
    to_screen(&white_coords, this_component->plot_scale);

    red_coords.fx = this_component->plot_scale->x_max;
    red_coords.fy = FULL_X_TO_Y_PTR ( exp ( log(red_coords.fx) * target_gamma ), this_component->cp_r );
    to_screen(&red_coords, this_component->plot_scale);

    green_coords.fx = this_component->plot_scale->x_max;
    green_coords.fy = FULL_X_TO_Y_PTR ( exp ( log(green_coords.fx) * target_gamma ), this_component->cp_g );
```



```
to_screen(&green_coords,this_component->plot_scale);

blue_coords.fx = this_component->plot_scale->x_max;
blue_coords.fy = FULL_X_TO_Y_PTR ( exp ( log(blue_coords.fx) * target_gamma ), this_component->cp_b );
to_screen(&blue_coords,this_component->plot_scale);

line_start.h = white_coords.sx + 20;
line_start.v = ( red_coords.sy + green_coords.sy + blue_coords.sy ) / 3;

RGBForeColor( &RGB_BLUE );
MoveTo( blue_coords.sx, blue_coords.sy );
LineTo( blue_coords.sx + 10, blue_coords.sy );
LineTo( blue_coords.sx + 10, line_start.v );
LineTo( blue_coords.sx + 20, line_start.v );

MoveTo( blue_coords.sx + 3, blue_coords.sy - 2 );
LineTo( blue_coords.sx + 1, blue_coords.sy );
LineTo( blue_coords.sx + 3, blue_coords.sy + 2 );

RGBForeColor( &RGB_GREEN );
MoveTo( green_coords.sx, green_coords.sy );
LineTo( green_coords.sx + 10, green_coords.sy );
LineTo( green_coords.sx + 10, line_start.v );
LineTo( green_coords.sx + 20, line_start.v );

MoveTo( green_coords.sx + 3, green_coords.sy - 2 );
LineTo( green_coords.sx + 1, green_coords.sy );
LineTo( green_coords.sx + 3, green_coords.sy + 2 );

RGBForeColor( &RGB_RED );
MoveTo( red_coords.sx, red_coords.sy );
LineTo( red_coords.sx + 10, red_coords.sy );
LineTo( red_coords.sx + 10, line_start.v );
LineTo( red_coords.sx + 20, line_start.v );

MoveTo( red_coords.sx + 3, red_coords.sy - 2 );
LineTo( red_coords.sx + 1, red_coords.sy );
LineTo( red_coords.sx + 3, red_coords.sy + 2 );

RGBForeColor( &RGB_WHITE );
MoveTo( line_start.h + 10, line_start.v + 7 );
DrawString("\pWhite Balance");
MoveTo( line_start.h + 10, line_start.v + 17 );
DrawString("\pAdjustment");
}

//

void draw_gamma_markers( struct calibration_component_info *this_component )
{
    struct coordinates these_coordinates;
    Point origin;
    char the_string[256];

    these_coordinates.fx = this_component->plot_scale->x_min;
    these_coordinates.fy = this_component->plot_scale->y_max;
    to_screen(&these_coordinates,this_component->plot_scale);
    origin.h = these_coordinates.sx;
    origin.v = these_coordinates.sy;

    RGBForeColor( &RGB_WHITE );
    MoveTo( origin.h + 5, origin.v + 12 );
    the_string[0] = 0;
    if( this_component->plot_target_perceptual )
        sprintf( the_string, "Target Gamma = Perceptual" );
    else
        sprintf( the_string, "Target Gamma = %5.3f", this_component->plot_target_gamma );
    DrawText( the_string, 0, strlen(the_string));
}

//
/* initialize_scale calculates all the nasty scaling factors */

void initialize_scale(struct scale_info *this_scale)
{
    this_scale->to_screen_x_scale = (this_scale->draw_rect.right - this_scale->draw_rect.left)/(this_scale->x_max - this_scale->x_min);
    this_scale->to_screen_x_offset = 0.5 + this_scale->draw_rect.left - this_scale->to_screen_x_scale * this_scale->x_min;

    this_scale->to_screen_y_scale = (this_scale->draw_rect.top - this_scale->draw_rect.bottom)/(this_scale->y_max - this_scale->y_min);
    this_scale->to_screen_y_offset = 0.5 + this_scale->draw_rect.bottom - this_scale->to_screen_y_scale * this_scale->y_min;

    this_scale->to_value_x_scale = 1.0/this_scale->to_screen_x_scale;
    this_scale->to_value_x_offset = this_scale->x_min - this_scale->to_value_x_scale * this_scale->draw_rect.left;
```

```
this_scale->to_value_y_scale = 1.0/this_scale->to_screen_y_scale;
this_scale->to_value_y_offset = this_scale->y_min - this_scale->to_value_y_scale *
this_scale->draw_rect.bottom;
}

//
/* to_screen converts to screen coordinates */

void to_screen(struct coordinates *my_coordinates,struct scale_info *this_scale)
{
    my_coordinates->sx = (short)(my_coordinates->fx * this_scale->to_screen_x_scale +
this_scale->to_screen_x_offset);
    my_coordinates->sy = (short)(my_coordinates->fy * this_scale->to_screen_y_scale +
this_scale->to_screen_y_offset);
}

//
/* to_value converts to a floating value */

void to_value(struct coordinates *my_coordinates,struct scale_info *this_scale)
{
    my_coordinates->fx = (float)my_coordinates->sx * this_scale->to_value_x_scale + this_scale->to_value_x_offset;
    my_coordinates->fy = (float)my_coordinates->sy * this_scale->to_value_y_scale + this_scale->to_value_y_offset;
}

//
/* rotate_pos_45 moves things up to the diagonal */

void rotate_pos_45(struct coordinates *my_coordinates)
{
    float x,y;

    x = my_coordinates->fx;
    y = my_coordinates->fy;

    my_coordinates->fx = 0.707106781187 * ( x - y );
    my_coordinates->fy = 0.707106781187 * ( x + y );
}

//

int lower_bulge(struct control_point_info *my_control_points)
{
    double indhi;
    double depfi;
    double d2hi;
    double max_ind;
    double x;

    if (my_control_points->count < 3)
        return 0;

    indhi = my_control_points->indep[1];
    depfi = my_control_points->dep[1];
    d2hi = my_control_points->d2[1];

    max_ind = sqrt(fabs(-6.0*depfi + 6.0*indhi + d2hi*(indhi*indhi)))/(sqrt(3.0)*sqrt(fabs(d2hi)));

    if(max_ind > indhi)
        return 0;

    x = 0.707106781187 * ( max_ind - interpolate_value(max_ind,my_control_points) );

    if ( x > 0.0 )
        return 0;
    else
        return 1;
}
```

```
/*
 * illus_out.h
 *
 *
 * Created by Robert Kay on Tue Dec 03 2002.
 * Copyright (c) 2002 __MyCompanyName__. All rights reserved.
 */

#include "cal_math.h"

int write_plot_file(struct calibration_component_info *);
void initialize_ill_scale(struct scale_info *);
void to_ill(struct ill_coord *, struct scale_info *);
```

```
/*
 *   illus_out.c
 *
 *   Created by Robert Kay on Tue Dec 03 2002.
 *   Copyright (c) 2002 __MyCompanyName__. All rights reserved.
 */

#include "illus_out.h"

int write_plot_file(struct calibration_component_info *this_component)
{
    static int plot_num=0;
    struct scale_info ill_scale;
    struct ill_coord these_coords;
    char filename[16];
    FILE *outfile;
    int i;
    float f_x, f_step;

    /* this sets the scale of the plot, */
    ill_scale.x_min = 0.0;
    ill_scale.x_max = 1.0;

    ill_scale.y_min = 0.0;
    ill_scale.y_max = 1.0;

    ill_scale.draw_rect.left   = 100;
    ill_scale.draw_rect.right  = 500;
    ill_scale.draw_rect.bottom = 100;
    ill_scale.draw_rect.top    = 500;

    initialize_ill_scale(&ill_scale);

    sprintf(filename, "mc%02d.ai", plot_num);

    outfile = fopen(filename, "w");

    if(outfile != 0)
    {
        /* necessary header stuff */
        fprintf(outfile, "%%!PS-Adobe-2.0\n");
        fprintf(outfile, "%%Creator: Adobe Illustrator(TM) 1.1\n");
        fprintf(outfile, "%%For: (Maskerade) {}\n");
        fprintf(outfile, "%%Title: (Plot%02d)\n", plot_num);
        fprintf(outfile, "%%CreationDate: (9/15/98) (1:05 PM)\n");
        fprintf(outfile, "%%DocumentProcessColors: Cyan Magenta Yellow\n");
        fprintf(outfile, "%%DocumentProcSets: Adobe_Illustrator_1.1 0 0\n");
        fprintf(outfile, "%%BoundingBox: %d %d %d\n", ill_scale.draw_rect.left, ill_scale.draw_rect.right, ill_scale.draw_rect.top);
        fprintf(outfile, "%%ColorUsage: Color\n");
        fprintf(outfile, "%%TemplateBox: %d %d %d\n", ill_scale.draw_rect.left, ill_scale.draw_rect.right, ill_scale.draw_rect.top);
        fprintf(outfile, "%%TileBox: %d %d %d\n", ill_scale.draw_rect.left, ill_scale.draw_rect.right, ill_scale.draw_rect.top);
        fprintf(outfile, "%%DocumentPreview: None\n");
        fprintf(outfile, "%%EndComments\n");
        fprintf(outfile, "%%EndProlog\n");
        fprintf(outfile, "%%BeginSetup\n");
        fprintf(outfile, "Adobe_Illustrator_1.1 begin\n");
        fprintf(outfile, "n\n");
        fprintf(outfile, "%%EndSetup\n");

        /* the actual plot */

        /* start the big group */
        fprintf(outfile, "u\n");

        /* plot frame */

        /* color and pen */
        fprintf(outfile, "%.3f %.3f %.3f %.3f K\n", 0.0, 0.0, 0.0, 1.0);
        fprintf(outfile, "%.3f w\n", 3.0);

        these_coords.fx_in = 0.0;
        these_coords.fy_in = 0.0;
        to_ill(&these_coords, &ill_scale);
        fprintf(outfile, "%.3f %.3f m\n", these_coords.fx_out, these_coords.fy_out);

        these_coords.fx_in = 0.0;
        these_coords.fy_in = 1.0;
        to_ill(&these_coords, &ill_scale);
        fprintf(outfile, "%.3f %.3f l\n", these_coords.fx_out, these_coords.fy_out);

        these_coords.fx_in = 1.0;
```

```
these_coords.fy_in = 1.0;
to_ill(&these_coords,&ill_scale);
fprintf(outfile,"%3f %3f 1\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = 1.0;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);
fprintf(outfile,"%3f %3f 1\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = 0.0;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);
fprintf(outfile,"%3f %3f 1\n",these_coords.fx_out,these_coords.fy_out);

fprintf(outfile,"s\n");

/* now the curve */

f_step = 1.41421356237/sqrt(ill_scale.to_screen_x_scale * ill_scale.to_screen_x_scale +
ill_scale.to_screen_y_scale * ill_scale.to_screen_y_scale);

/* color and pen */
fprintf(outfile,"%3f %3f %3f %3f K\n",0.0,0.55,1.0,0.0);
fprintf(outfile,"%3f w\n",2.0);

these_coords.fx_in = 0.0;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);
fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

for(f_x=0.0;f_x<=1.41421356237;f_x+=f_step)
{
    these_coords.fx_in = f_x;
    these_coords.fy_in = interpolate_value(these_coords.fx_in,this_component->cp_cur);
    rotate_pos_45((struct coordinates *)&these_coords);
    to_ill(&these_coords,&ill_scale);
    fprintf(outfile,"%3f %3f 1\n",these_coords.fx_out,these_coords.fy_out);
}

fprintf(outfile,"S\n");

/* now draw the test point lines */

/* start the lines group */
fprintf(outfile,"u\n");

/* color */
/* color and pen */
fprintf(outfile,"%3f %3f %3f %3f K\n",0.62,0.0,0.52,0.0);
fprintf(outfile,"%3f w\n",0.75);

for(i=0;i<this_component->test_point_count;i++)
{
    if(this_component->test_points[i].point_status != 1)
    {
        /* draw a line to show a possible calibration point */

        these_coords.fx_in = this_component->test_points[i].x;
        these_coords.fy_in = 0.0;
        to_ill(&these_coords,&ill_scale);
        fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);
        these_coords.fx_in = this_component->test_points[i].x;
        these_coords.fy_in = 1.0;
        to_ill(&these_coords,&ill_scale);
        fprintf(outfile,"%3f %3f 1\n",these_coords.fx_out,these_coords.fy_out);
        fprintf(outfile,"S\n");
    }
}

/* end the lines group */
fprintf(outfile,"U\n");

/* start the control lines group */
fprintf(outfile,"u\n");

/* color and pen */
fprintf(outfile,"%3f %3f %3f %3f K\n",0.4353,0.5333,0.0,0.0);
fprintf(outfile,"%3f w\n",0.75);

for(i=0;i<this_component->test_point_count;i++)
{
    if(this_component->test_points[i].point_status == 1)
    {
```

```
/* draw a line to show a calibration point */

these_coords.fx_in = this_component->test_points[i].x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);
fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);
these_coords.fx_in = this_component->test_points[i].x;
these_coords.fy_in = 1.0;
to_ill(&these_coords,&ill_scale);
fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");
}

/* end the control lines group */
fprintf(outfile,"U\n");

/* start the control X group */
fprintf(outfile,"u\n");

/* color and pen */
fprintf(outfile,"%3f %3f %3f %3f K\n",0.0,0.0,0.0,1.0);
fprintf(outfile,"%3f w\n",0.75);

for(i=0;i<this_component->test_point_count;i++)
{
    if(this_component->test_points[i].point_status == 1)
    {
        /* draw an X */
        /* start this X group */
        fprintf(outfile,"u\n");

        these_coords.fx_in = this_component->test_points[i].x - 0.01;
        these_coords.fy_in = this_component->test_points[i].y + 0.01;
        to_ill(&these_coords,&ill_scale);

        fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

        these_coords.fx_in = this_component->test_points[i].x + 0.01;
        these_coords.fy_in = this_component->test_points[i].y - 0.01;
        to_ill(&these_coords,&ill_scale);

        fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);

        fprintf(outfile,"S\n");

        these_coords.fx_in = this_component->test_points[i].x - 0.01;
        these_coords.fy_in = this_component->test_points[i].y - 0.01;
        to_ill(&these_coords,&ill_scale);

        fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

        these_coords.fx_in = this_component->test_points[i].x + 0.01;
        these_coords.fy_in = this_component->test_points[i].y + 0.01;
        to_ill(&these_coords,&ill_scale);

        fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);

        fprintf(outfile,"S\n");

        /* end this X group */
        fprintf(outfile,"U\n");
    }
}

/* end the control X group */
fprintf(outfile,"U\n");

/* draw active line */

/* start this -> group */
fprintf(outfile,"u\n");

/* color and pen */
fprintf(outfile,"%3f %3f %3f %3f K\n",0.0196,0.8666,0.3216,0.0);
fprintf(outfile,"%3f w\n",1.0);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].x - 0.01;
these_coords.fy_in = -0.01;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].x;
these_coords.fy_in = 0.0;
```

```
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

these_coords.fx_in = this_component->test_points[this_component->nearest_point].x + 0.01;
these_coords.fy_in = -0.01;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

these_coords.fx_in = this_component->test_points[this_component->nearest_point].x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].x;
these_coords.fy_in = -0.05;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

/* end this -> group */
fprintf(outfile,"U\n");

/* and it's parents */

/* start the parents group */
fprintf(outfile,"u\n");

/* start this -> group */
fprintf(outfile,"u\n");

/* color and pen */
fprintf(outfile,"%3f %3f %3f %3f K\n",0.698,0.1372,0.2706,0.0078);
fprintf(outfile,"%3f w\n",1.0);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.a_x - 0.01;
these_coords.fy_in = -0.01;
to_ill(&these_cords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.a_x;
these_coords.fy_in = 0.0;
to_ill(&these_cords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.a_x + 0.01;
these_coords.fy_in = -0.01;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.a_x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.a_x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.a_x;
these_coords.fy_in = -0.03;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

/* end this -> group */
```

```
fprintf(outfile,"U\n");

/* start this -> group */
fprintf(outfile,"u\n");

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.b_x - 0.01;
these_coords.fy_in = -0.01;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.b_x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.b_x + 0.01;
these_coords.fy_in = -0.01;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.b_x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");


these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.b_x;
these_coords.fy_in = 0.0;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f m\n",these_coords.fx_out,these_coords.fy_out);

these_coords.fx_in = this_component->test_points[this_component->nearest_point].parents.b_x;
these_coords.fy_in = -0.03;
to_ill(&these_coords,&ill_scale);

fprintf(outfile,"%3f %3f l\n",these_coords.fx_out,these_coords.fy_out);
fprintf(outfile,"S\n");

/* end this -> group */
fprintf(outfile,"U\n");

/* end the parents group */
fprintf(outfile,"U\n");

/* end the big group */
fprintf(outfile,"U\n");

/* trailer stuff */
fprintf(outfile,"%%%PageTrailer\n");
fprintf(outfile,"%%%Trailer\n");
fprintf(outfile," E end\n");
fprintf(outfile,"%%%EOF\n");

fclose(outfile);
//fsetfileinfo(filename,'ART5','TEXT');

plot_num++;

return 1;
}

return 0;
}

void initialize_ill_scale(struct scale_info *this_scale)
{
    this_scale->to_screen_x_scale = (this_scale->draw_rect.right - this_scale->draw_rect.left)/(this_scale->x_max -
this_scale->x_min);
    this_scale->to_screen_x_offset = this_scale->draw_rect.left - this_scale->to_screen_x_scale *
this_scale->x_min;

    this_scale->to_screen_y_scale = (this_scale->draw_rect.top - this_scale->draw_rect.bottom)/(this_scale->y_max -
this_scale->y_min);
```



```
        this_scale->to_screen_y_offset = this_scale->draw_rect.bottom - this_scale->to_screen_y_scale *
this_scale->y_min;
    }

void to_ill(struct ill_coord *my_coordinates, struct scale_info *this_scale)
{
    my_coordinates->fx_out = my_coordinates->fx_in * this_scale->to_screen_x_scale +
this_scale->to_screen_x_offset;
    my_coordinates->fy_out = my_coordinates->fy_in * this_scale->to_screen_y_scale +
this_scale->to_screen_y_offset;
}
```

```
//  
//  
//  
#ifndef __o_asst_dialog__  
#define __o_asst_dialog__  
  
#include "o_base_dialog.h"  
#include "my_controls.h"  
  
//class o_asst_dialog;  
  
#include "globals.h"  
  
#include "o_base_asst_pane.h"  
  
#include "o_intro_pane.h"  
#include "o_new_or_adjust_pane.h"  
#include "o_display_type_pane.h"  
#include "o_control_type_pane.h"  
#include "o_adjust_display_pane.h"  
#include "o_phosphors_pane.h"  
#include "o_black_level_pane.h"  
#include "o_response_pane.h"  
#include "o_white_point_pane.h"  
#include "o_gamma_target_pane.h"  
#include "o_save_profile_pane.h"  
  
//extern struct cal_globals;  
  
enum  
{  
    kAssistantDialogDLOG    = 128  
};  
  
enum  
{  
    kAssistantAnchorText    = 1,  
    kAssistantHeader        = 2,  
    kAssistantFooter        = 3,  
    kAssistantLeftButton    = 4,  
    kAssistantRightButton   = 5,  
    kAssistantPaneNumber    = 6,  
    kAssistantPaneTitle     = 7,  
    kAssistantAppName       = 8,  
    kAssistantIcon          = 9,  
    kAssistantWhyButton     = 10,  
    kAssistantCautionButton = 11,  
    kAssistantTechInfoButton = 12  
};  
  
enum  
{  
    kIntroPane              = 1,  
    kNewOrAdjustPane,  
    kDisplayTypePane,  
    kControlTypePane,  
    kAdjustDisplayPane,  
    kBlackLevelPane,  
    kGammaPane,  
    kWhitePointPane,  
    kGammaTargetPane,  
    kPhosphorsPane,  
    kSaveProfilePane,  
    kEndPane,  
  
    kWhyExpertModePane,  
    kWhyDisplayTypePane,  
    kWhyBlackLevelPane,  
    kCautionLCDBlackLevelPane,  
    kWhyResponsePane,  
    kCautionLCDResponsePane,  
    kWhyPhosphorTypePane,  
    kCautionGenericPhosphorPane,  
    kWhyTargetResponsePane,  
    kWhyControlTypePane,  
    kWhyWhitePointPane,  
  
    kLastValidPane  
};  
  
//
```

```
class o_asst_dialog : public o_base_dialog
{
public:
    // constructors & destructors
        o_asst_dialog ( short, struct cal_globals * );
        ~o_asst_dialog ();

    // drawing and clicking
    void do_Handle_Item_Hit ( short );
    Boolean do_Handle_Key_Down ( EventRecord * );
    void do_Key_Down_Post_Processing ();
// Boolean do_OK_To_Close();
    void do_Draw ();
    void do_Idle ();

protected:

private:
    void do_Switch_Pane ( short pane_index );
    void do_Why_Mode ();
    void do_Caution_Mode ();
    void do_Normal_Mode ();

    o_base_asst_pane *pane_ptr;
    short pane_number;
    struct cal_globals *globals;
};

#endif /* __o_asst_dialog__ */
```

```
// _____  
// _____ ©1998-2001 bergdesign inc.  
// _____  
  
#include "o_asst_dialog.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
o_asst_dialog::o_asst_dialog ( short dlog_id, struct cal_globals *cal_globals ) : o_base_dialog ( dlog_id )  
{  
    DEBUG_PRINT("Entered o_asst_dialog constructor");  
  
    pane_ptr = NULL;  
    pane_number = kIntroPane;  
    globals = cal_globals;  
  
    TextFont ( applFont );  
    TextSize ( 10 );  
  
    // do_Set_Bevel_Button_Graphic_Alignment ( dialog_ref, kAssistantCautionButton, kControlBevelButtonAlignCenter, 0,  
    0 );  
    // do_Set_Bevel_Button_Text_Placement ( dialog_ref, kAssistantCautionButton,  
    kControlBevelButtonPlaceToRightOfGraphic, 0 );  
    // do_Set_Bevel_Button_Text_Alignment ( dialog_ref, kAssistantCautionButton,  
    kControlBevelButtonPlaceToRightOfGraphic, 0 );  
  
    do_Switch_Pane ( kIntroPane );  
  
    DEBUG_PRINT("Left o_asst_dialog constructor");  
}  
  
// _____  
o_asst_dialog::~o_asst_dialog ()  
{  
    DEBUG_PRINT("Entered o_asst_dialog destructor");  
  
    if ( pane_ptr )  
        delete pane_ptr;  
  
    DEBUG_PRINT("Left o_asst_dialog destructor");  
}  
  
// _____  
#pragma mark -  
  
// _____  
void o_asst_dialog::do_Handle_Item_Hit ( short item_hit )  
{  
    switch ( item_hit )  
    {  
        case kAssistantLeftButton:  
        {  
            if ( pane_number > kIntroPane )  
            {  
                pane_number--;  
            }  
  
            do_Switch_Pane ( pane_number );  
            break;  
        }  
        case kAssistantRightButton:  
        {  
            if ( pane_number < kEndPane )  
            {  
                pane_number++;  
            }  
  
            do_Switch_Pane ( pane_number );  
            break;  
        }  
        case kAssistantWhyButton:  
        {  
            switch ( pane_number )  
            {  
                case kDisplayTypePane:  
                    pane_number = kWhyDisplayTypePane;  
                    break;  
                case kWhyDisplayTypePane:  
                    pane_number = kDisplayTypePane;  
                    break;  
            }  
        }  
    }  
}
```

```
        case kControlTypePane:
            pane_number = kWhyControlTypePane;
            break;
        case kWhyControlTypePane:
            pane_number = kControlTypePane;
            break;
        case kBlackLevelPane:
            pane_number = kWhyBlackLevelPane;
            break;
        case kWhyBlackLevelPane:
            pane_number = kBlackLevelPane;
            break;
        case kGammaPane:
            pane_number = kWhyResponsePane;
            break;
        case kWhyResponsePane:
            pane_number = kGammaPane;
            break;
        case kPhosphorsPane:
            pane_number = kWhyPhosphorTypePane;
            break;
        case kWhyPhosphorTypePane:
            pane_number = kPhosphorsPane;
            break;
        case kGammaTargetPane:
            pane_number = kWhyTargetResponsePane;
            break;
        case kWhyTargetResponsePane:
            pane_number = kGammaTargetPane;
            break;
        default:
            break;
    }

    do_Switch_Pane ( pane_number );
    break;
}
case kAssistantCautionButton:
{
    switch ( pane_number )
    {
        case kBlackLevelPane:
            pane_number = kCautionLCDBlackLevelPane;
            break;
        case kCautionLCDBlackLevelPane:
            pane_number = kBlackLevelPane;
            break;
        case kGammaPane:
            pane_number = kCautionLCDResponsePane;
            break;
        case kCautionLCDResponsePane:
            pane_number = kGammaPane;
            break;
        case kPhosphorsPane:
            pane_number = kCautionGenericPhosphorPane;
            break;
        case kCautionGenericPhosphorPane:
            pane_number = kPhosphorsPane;
            break;
        default:
            break;
    }

    do_Switch_Pane ( pane_number );
    break;
}
case kAssistantTechInfoButton:
{
    break;
}
case kAssistantIcon:
{
    do_Dump_Control_Hierarchy ( GetDialogWindow(dialog_ref) );
    break;
}
default:
{
    pane_ptr->do_Item_Hit ( item_hit );
    if ( pane_number == kSaveProfilePane && globals->create_profile == true )
    {
        do_Activate_DItem ( dialog_ref, kAssistantRightButton, true );

        // When should we automatically switch to the finished pane?
        if( cmNoProfileBase == globals->chosen_profile_loc.locType )
            do_Switch_Pane ( kEndPane );
    }
    break;
}
}
```

```
}  
}  
  
//  
  
Boolean o_asst_dialog::do_Handle_Key_Down ( EventRecord *event )  
{  
    Boolean          key_was_handled = false;  
    OSStatus         err = noErr;  
    ControlHandle     control = NULL;  
    ControlPartCode   part_code;  
    char              the_key = event->message & charCodeMask;  
  
    err = GetKeyboardFocus ( GetDialogWindow(dialog_ref), &control );  
  
    if ( err == errNoRootControl )  
    {  
        key_was_handled = false;  
    }  
    else if ( control == NULL )  
    {  
        if ( the_key == kTabCharCode )  
        {  
            short      num_items = CountDITL ( (DialogRef)dialog_ref );  
            short       i;  
            unsigned long features = 0;  
  
            if ( num_items > 0 )  
            {  
                for ( i = 1; i <= num_items; i++ )  
                {  
                    GetDialogItemAsControl ( dialog_ref, i, &control );  
  
                    if ( control )  
                    {  
                        err = GetControlFeatures ( control, &features );  
  
                        if ( !err )  
                        {  
                            if ( ( features & kControlSupportsFocus ) && ( features & kControlGetsFocusOnClick ) )  
                                err = SetKeyboardFocus ( GetDialogWindow(dialog_ref), control, kControlEditTextPart );  
  
                            if ( !err )  
                            {  
                                key_was_handled = true;  
                                break;  
                            }  
                            else  
                            {  
                                continue;  
                            }  
                        }  
                    }  
                }  
            }  
        }  
        else  
        {  
            key_was_handled = true;  
        }  
    }  
    else if ( the_key == kLeftArrowCharCode )  
    {  
        do_Handle_Item_Hit ( kAssistantLeftButton );  
    }  
    else if ( the_key == kRightArrowCharCode )  
    {  
        do_Handle_Item_Hit ( kAssistantRightButton );  
    }  
    else  
    {  
        key_was_handled = false;  
    }  
}  
else  
{  
    if ( ( the_key == kTabCharCode ) && ( event->modifiers & shiftKey ) )  
    {  
        ReverseKeyboardFocus ( GetDialogWindow(dialog_ref) );  
    }  
    else if ( the_key == kTabCharCode )  
    {  
        AdvanceKeyboardFocus ( GetDialogWindow(dialog_ref) );  
    }  
    else  
    {  
    }
```

```
        part_code = HandleControlKey ( control,
                                        (short)(event->message & keyCodeMask),
                                        (short)the_key,
                                        (short)event->modifiers);

        if ( part_code )
        {
            key_was_handled = true;
            do_Key_Down_Post_Processing ();
        }
    }

    return ( key_was_handled );
}

//
void o_asst_dialog::do_Key_Down_Post_Processing ()
{
    if ( pane_ptr )
        pane_ptr->do_Key_Down_Post_Processing();
}

//
/*
Boolean o_asst_dialog::do_OK_To_Close()
{
    Boolean    ok_to_close = true;
    Sint16     answer = kAlertStdAlertCancelButton;

    if( ( globals->create_profile != true ) && ( globals->black_level_complete || globals->response_complete ) )
    {
        // We ask the user if they wish to quit.
        answer = do_Two_Button_Alert ( kAlertCautionAlert,
                                      "\pAre you sure you want to quit SuperCal?",
                                      "\pYou will lose any measurements that you have made.",
                                      "\pQuit",
                                      "\pCancel" );

        if( kAlertStdAlertOKButton == answer )
            ok_to_close = true;
        else
            ok_to_close = false;
    }

    return( ok_to_close );
}
*/
//
void o_asst_dialog::do_Draw ()
{
    // When we override this, we don't need to worry about port stuff.
    // We just assume everything is ok and start drawing.

    DEBUG_PRINT("Entered o_asst_dialog::do_Draw()");

    // We are simply adding a method to piggyback on top of activate
    // and update events so that we can change button titles and
    // other things in our dialog panes.

    if ( pane_ptr )
        pane_ptr->do_Update();

    // Need to call our base class drawing routine so it will
    // do it's drawing functions too.
    o_base_dialog::do_Draw();

    DEBUG_PRINT("Left o_asst_dialog::do_Draw()");
}

//
void o_asst_dialog::do_Idle ()
{
    if ( pane_ptr )
        pane_ptr->do_Idle();
}

//
void o_asst_dialog::do_Switch_Pane ( short pane_number )
{
    if ( pane_number < kIntroPane || pane_number > ( kLastValidPane - 1 ) )
        return;

    if ( pane_ptr )
    {

```

```
    delete pane_ptr;  
    pane_ptr = NULL;  
    CompactMem(maxSize);  
}  
  
switch ( pane_number )  
{  
    case kIntroPane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pIntroduction", false );  
        HideDialogItem ( dialog_ref, kAssistantWhyButton );  
        HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        // pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kIntroPaneDITL,  
        kBaseAsstPaneAppendMode, globals );  
        pane_ptr = new o_intro_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;  
  
    case kNewOrAdjustPane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pNew or Edit", false );  
        HideDialogItem ( dialog_ref, kAssistantWhyButton );  
        HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        do_Normal_Mode ();  
        pane_ptr = new o_new_or_adjust_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;  
  
    case kDisplayTypePane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pDisplay Type", false );  
        ShowDialogItem ( dialog_ref, kAssistantWhyButton );  
        HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        do_Normal_Mode ();  
        pane_ptr = new o_display_type_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;  
  
    case kControlTypePane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pDisplay Controls", false );  
        ShowDialogItem ( dialog_ref, kAssistantWhyButton );  
        HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        do_Normal_Mode ();  
        pane_ptr = new o_control_type_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;  
  
    case kAdjustDisplayPane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pControl Adjustment", false );  
        HideDialogItem ( dialog_ref, kAssistantWhyButton );  
        HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        do_Normal_Mode ();  
        pane_ptr = new o_adjust_display_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;  
  
    case kBlackLevelPane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pBlack Level Measurement", false );  
        ShowDialogItem ( dialog_ref, kAssistantWhyButton );  
        if ( globals->display_type == kDisplayTypeLCD )  
            ShowDialogItem ( dialog_ref, kAssistantCautionButton );  
        else  
            HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        do_Normal_Mode ();  
        pane_ptr = new o_black_level_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;  
  
    case kGammaPane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pResponse Measurement", false );  
        ShowDialogItem ( dialog_ref, kAssistantWhyButton );  
        if ( globals->display_type == kDisplayTypeLCD )  
            ShowDialogItem ( dialog_ref, kAssistantCautionButton );  
        else  
            HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        do_Normal_Mode ();  
        pane_ptr = new o_response_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;  
  
    case kWhitePointPane:  
        do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pWhite Balance", false );  
        HideDialogItem ( dialog_ref, kAssistantWhyButton );  
        if ( globals->display_type == kDisplayTypeLCD )  
            HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        else  
            HideDialogItem ( dialog_ref, kAssistantCautionButton );  
        HideDialogItem ( dialog_ref, kAssistantTechInfoButton );  
        do_Normal_Mode ();  
        pane_ptr = new o_white_point_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );  
        break;
```



```
case kGammaTargetPane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pTarget Response", false );
    ShowDialogItem ( dialog_ref, kAssistantWhyButton );
    HideDialogItem ( dialog_ref, kAssistantCautionButton );
    HideDialogItem ( dialog_ref, kAssistantTechInfoButton );
    do_Normal_Mode ();
    pane_ptr = new o_gamma_target_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );
    break;

case kPhosphorsPane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pDisplay Colors", false );
    ShowDialogItem ( dialog_ref, kAssistantWhyButton );
    HideDialogItem ( dialog_ref, kAssistantCautionButton );
    HideDialogItem ( dialog_ref, kAssistantTechInfoButton );
    do_Normal_Mode ();
    pane_ptr = new o_phosphors_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );
    break;

case kSaveProfilePane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pSave Your Profile", false );
    HideDialogItem ( dialog_ref, kAssistantWhyButton );
    HideDialogItem ( dialog_ref, kAssistantCautionButton );
    HideDialogItem ( dialog_ref, kAssistantTechInfoButton );
    do_Normal_Mode ();
    if ( globals->create_profile == false )
    {
        do_Activate_DItem ( dialog_ref, kAssistantRightButton, false );
    }
    pane_ptr = new o_save_profile_pane ( dialog_ref, CountDITL ( dialog_ref ), globals );
    break;

case kEndPane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pFinished", false );
    HideDialogItem ( dialog_ref, kAssistantWhyButton );
    HideDialogItem ( dialog_ref, kAssistantCautionButton );
    HideDialogItem ( dialog_ref, kAssistantTechInfoButton );
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kEndPaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kWhyDisplayTypePane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pWhy Select A Display Type?", false );
    do_Why_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kWhyDisplayTypePaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kWhyControlTypePane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pWhy Select the Types of Controls?",
false );
    do_Why_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kWhyControlTypePaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kWhyBlackLevelPane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pWhy Measure the Black Level?",
false );
    do_Why_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kWhyBlackLevelPaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kWhyResponsePane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pWhy Measure the Display Response?",
false );
    do_Why_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kWhyResponsePaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kWhyPhosphorTypePane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pWhy Select the Display Colors?",
false );
    do_Why_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kWhyPhosphorTypePaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kWhyTargetResponsePane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pWhy Set the Target Response?",
false );
    do_Why_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kWhyTargetResponsePaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;
```

```
case kCautionLCDBlackLevelPane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pLCD Measurement Caution", false )
    do_Caution_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kCautionLCDBlackLevelPaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kCautionLCDResponsePane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pLCD Measurement Caution", false )
    do_Caution_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kCautionLCDResponsePaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

case kCautionGenericPhosphorPane:
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneTitle, "\pGeneric Phosphor Type Caution",
false );
    do_Caution_Mode();
    pane_ptr = new o_base_asst_pane ( dialog_ref, CountDITL ( dialog_ref ), kCautionGenericPhosphorPaneDITL,
kBaseAsstPaneAppendMode, globals );
    break;

default:
    break;
}

if ( pane_number >= kIntroPane && pane_number <= kEndPane )
{
    Str255 pane_number_label;
    pane_number_label[0] = 0;
    do_p_strerrcat( pane_number_label, pane_number );
    do_p_strcat( pane_number_label, "\p of " );
    do_p_strerrcat( pane_number_label, kEndPane );
    do_Set_Text_Of_DItem_As_PString ( dialog_ref, kAssistantPaneNumber, pane_number_label, false );
}

if ( pane_number == kIntroPane )
{
    HideDialogItem ( dialog_ref, kAssistantLeftButton );
}
else
{
    ShowDialogItem ( dialog_ref, kAssistantLeftButton );
}

if ( pane_number == kEndPane )
{
    HideDialogItem ( dialog_ref, kAssistantRightButton );
}
else
{
    ShowDialogItem ( dialog_ref, kAssistantRightButton );
}
}

//
void o_asst_dialog::do_Why_Mode ()
{
    do_Set_Title_Of_DItem ( dialog_ref, kAssistantWhyButton, "\pBack" );
    ShowDialogItem ( dialog_ref, kAssistantWhyButton );
    do_Activate_DItem ( dialog_ref, kAssistantWhyButton, true );

    if ( do_Is_DItem_Visible ( dialog_ref, kAssistantCautionButton ) )
        do_Activate_DItem ( dialog_ref, kAssistantCautionButton, false );

    if ( do_Is_DItem_Visible ( dialog_ref, kAssistantTechInfoButton ) )
        do_Activate_DItem ( dialog_ref, kAssistantTechInfoButton, false );

    do_Activate_DItem ( dialog_ref, kAssistantLeftButton, false );
    do_Activate_DItem ( dialog_ref, kAssistantRightButton, false );
}

//
void o_asst_dialog::do_Caution_Mode ()
{
    ControlButtonContentInfo button_info;

    // the why button
    if ( do_Is_DItem_Visible ( dialog_ref, kAssistantWhyButton ) )
        do_Activate_DItem ( dialog_ref, kAssistantWhyButton, false );

    // the caution button
    button_info.contentType = kControlContentTextOnly;

    do_Set_Bevel_Button_Content_Info ( dialog_ref, kAssistantCautionButton, &button_info );
}
```

```
do_Set_Title_Of_DItem ( dialog_ref, kAssistantCautionButton, "\pBack" );

// the tech info button
if ( do_Is_DItem_Visible ( dialog_ref, kAssistantTechInfoButton ) )
    do_Activate_DItem ( dialog_ref, kAssistantTechInfoButton, false );

// the navigation buttons
do_Activate_DItem ( dialog_ref, kAssistantLeftButton, false );
do_Activate_DItem ( dialog_ref, kAssistantRightButton, false );
}

//
void o_asst_dialog::do_Normal_Mode ()
{
    ControlButtonContentInfo    button_info;

    // the why button
    do_Set_Title_Of_DItem ( dialog_ref, kAssistantWhyButton, "\pWhy?" );
    if ( do_Is_DItem_Visible ( dialog_ref, kAssistantWhyButton ) )
        do_Activate_DItem ( dialog_ref, kAssistantWhyButton, true );

    // the caution button
    button_info.contentType = kControlContentCIconRes;
    button_info.u.resID = 134;

    do_Set_Bevel_Button_Content_Info ( dialog_ref, kAssistantCautionButton, &button_info );
    do_Set_Title_Of_DItem ( dialog_ref, kAssistantCautionButton, "\p" );
    if ( do_Is_DItem_Visible ( dialog_ref, kAssistantCautionButton ) )
        do_Activate_DItem ( dialog_ref, kAssistantCautionButton, true );

    // the tech info button
    if ( do_Is_DItem_Visible ( dialog_ref, kAssistantTechInfoButton ) )
        do_Activate_DItem ( dialog_ref, kAssistantTechInfoButton, true );

    // the navigation buttons
    do_Activate_DItem ( dialog_ref, kAssistantLeftButton, true );
    do_Activate_DItem ( dialog_ref, kAssistantRightButton, true );
}
```

```
//  
// _____ ©1998-2001 bergdesign inc.  
//  
  
#ifndef __o_black_window__  
#define __o_black_window__  
  
#include "o_base_window.h"  
  
#include "my_menus.h"  
#include "globals.h"  
  
// _____  
  
class o_black_window : public o_base_window  
{  
public:  
  
    ControlHandle    done_button;  
  
    // constructors & destructors  
    o_black_window ( Rect *, Boolean, WindowAttributes, ThemeBrush, WindowRef, struct cal_globals *  
        -o_black_window ();  
  
    Boolean          do_Handle_Content_Click ( EventRecord * );  
  
protected:  
  
private:  
    struct cal_globals    *globals;  
  
};  
  
#endif /* __o_black_window__ */
```

```
// _____  
//                                     ©1998-2001 bergdesign inc.  
// _____  
  
#include "o_black_window.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
o_black_window::o_black_window ( Rect *bounds, Boolean visible, WindowAttributes attributes,  
                                ThemeBrush brush, WindowRef behind, struct cal_globals *cal_globals )  
    : o_base_window ( bounds, NULL, NULL, visible, attributes, brush, behind )  
{  
    Rect    done_button_rect;  
    Point   center;  
  
    DEBUG_PRINT("Entered o_black_window constructor");  
  
    globals = cal_globals;  
  
    do_Set_Title ( "\pWhite/Black Window" );  
    do_Show_Menu_Bar ( false );  
  
    center.h = ( bounds->right - bounds->left ) / 2;  
    center.v = ( bounds->bottom - bounds->top ) / 2;  
  
    SetRect ( &done_button_rect, center.h - 30, center.v - 10, center.h + 30, center.v + 10 );  
    done_button = NewControl ( window_ref, &done_button_rect, "\pDone", true, 0, 0, 1,  
(short)kControlPushButtonProc, (long)this );  
  
    DEBUG_PRINT("Left o_black_window constructor");  
}  
  
// _____  
  
o_black_window::~o_black_window ()  
{  
    DEBUG_PRINT("Entered o_black_window destructor");  
  
    if ( done_button )  
        DisposeControl ( done_button );  
  
    do_Show_Menu_Bar ( true );  
  
    DEBUG_PRINT("Left o_black_window destructor");  
}  
  
// _____  
  
#pragma mark -  
  
// _____  
  
Boolean o_black_window::do_Handle_Content_Click ( EventRecord *event )  
{  
    ControlHandle    control = NULL;  
    ControlPartCode  part_code;  
    Boolean          click_handled = false;  
    Point            where;  
  
    where = event->where;  
    GlobalToLocal ( &where ); // the current port must be correct or GlobalToLocal won't work right  
  
    control = FindControlUnderMouse ( where, window_ref, &part_code );  
  
    if ( control )  
    {  
        DEBUG_VAR_PRINT("FindControlUnderMouse() found control = %#010x",control);  
        DEBUG_EXTRA_VAR_PRINT(", part_code = %d",part_code);  
  
        if ( part_code != kControlNoPart && IsControlVisible(control) && IsControlActive(control) )  
        {  
            part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );  
            DEBUG_VAR_PRINT("HandleControlClick() returned part_code = %d",part_code);  
  
            if ( part_code )  
            {  
                if ( control == done_button )  
                {  
                    globals->black_window = NULL;  
                    click_handled = true;  
                    delete this;  
                }  
            }  
        }  
    }  
}
```

```
    }  
    return ( click_handled );  
}
```

```
//  
// _____  
// _____  
// _____  
  
#ifndef __o_black_level_window__  
#define __o_black_level_window__  
  
#include "o_base_window.h"  
// #include "o_cal_slider.h"  
  
#include "my_menus.h"  
#include "globals.h"  
#include "cal_math.h"  
#include "gamma_utils.h"  
  
// _____  
  
class o_black_level_window : public o_base_window  
{  
public:  
  
    // constructors & destructors  
    o_black_level_window ( Rect *, Boolean, WindowAttributes, ThemeBrush, WindowRef, struct  
cal_globals * );  
    ~o_black_level_window ();  
  
    Boolean    do_Handle_Content_Click ( EventRecord * );  
    void        do_Draw ();  
  
protected:  
  
    static pascal void    adjust_amb(ControlHandle,short);  
    void                do_Init_Black_Level ();  
  
private:  
  
    // o_cal_slider    *h_slider;  
    // o_cal_slider    *v_slider;  
    struct cal_globals    *globals;  
    struct slider_info    amb_slider;  
    ControlHandle        next_button;  
    ControlHandle        cancel_button;  
    PaletteHandle        the_palette;  
    int                    which_color;  
    PicHandle            pattern_pict;  
    PicHandle            solid_pict;  
};  
  
#endif /* __o_black_level_window__ */
```

```
// _____  
//                                     ©1998-2001 bergdesign inc.  
// _____  
  
#include "o_black_level_window.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
o_black_level_window::o_black_level_window ( Rect *bounds, Boolean visible, WindowAttributes attributes,  
                                              ThemeBrush brush, WindowRef behind, struct cal_globals *cal_globals )  
    : o_base_window ( bounds, NULL, NULL, visible, attributes, brush, behind )  
{  
    DEBUG_PRINT("Entered o_black_level_window constructor");  
  
    globals = cal_globals;  
  
    do_Set_Title ( "\pCalibration" );  
    do_Show_Menu_Bar ( false );  
  
    OSStatus err = linear_gamma_to_dev ( globals->this_component.this_dev_info );  
    do_Alert_If_Error ( "\pAn error occurred loading a linear gamma table.", err );  
  
    the_palette = NULL;  
    // initialize_palette ( (WindowPtr>window_ref, &the_palette );  
  
    pattern_pict = NULL;  
    solid_pict = NULL;  
  
    which_color = 1;  
  
    Rect port_rect = do_Get_Port_Rect();  
  
    SetRect ( &(amb_slider.slider_rect), port_rect.right - 24, port_rect.top + 4, port_rect.right - 8,  
port_rect.bottom - 64 );  
    amb_slider.my_window = (WindowPtr>window_ref;  
    amb_slider.live_function = (ProcPtr)(o_black_level_window::adjust_amb);  
    amb_slider.data_pointer = &(globals->this_component);  
    amb_slider.slider_min = 0.0;  
    amb_slider.slider_max = AMB_MAX;  
    initialize_slider(&amb_slider);  
    globals->this_component.amb_slider = &amb_slider;  
    ShowControl(amb_slider.this_control);  
    ActivateControl(amb_slider.this_control);  
  
    Rect button_rect = {0,0,0,0};  
    SetRect ( &button_rect, port_rect.right - 122, port_rect.bottom - 32, port_rect.right - 12, port_rect.bottom -  
12 );  
    next_button = NewControl( window_ref, &button_rect, "\pNext Color", false, 0, 0, 1,  
(short)kControlPushButtonProc, (long)this );  
    ShowControl( next_button );  
  
    // OffsetRect ( &button_rect, -92, 0 );  
    SetRect ( &button_rect, port_rect.left + 12, port_rect.bottom - 32, port_rect.left + 92, port_rect.bottom - 12  
);  
    cancel_button = NewControl( window_ref, &button_rect, "\pCancel", false, 0, 0, 1,  
(short)kControlPushButtonProc, (long)this );  
    ShowControl( cancel_button );  
  
    // globals->this_component.my_window = &((WindowPtr>window_ref);  
    // globals->this_component.next_button = &next_button;  
  
    do_Init_Black_Level ();  
  
    pattern_pict = GetPicture ( kCalibrationPattern_1x1 );  
    solid_pict = GetPicture ( kCalibrationSolid_008 );  
    // solid_pict = GetPicture ( kCalibrationGradient );  
  
    if ( !pattern_pict || !solid_pict )  
    {  
        do_One_Button_Alert ( kAlertStopAlert,  
                             "\pThere was not enough memory to load the calibration picture.",  
                             "\pPlease use Get Info and increase the preferred memory for Monitors & Sound by 25  
                             "\pOK" );  
    }  
  
    // do_Force_Update();  
    do_Force_Draw();  
  
    DEBUG_PRINT("Left o_black_level_window constructor");  
}  
  
// _____  
  
o_black_level_window::~o_black_level_window ()
```



```
{
    DEBUG_PRINT("Entered o_black_level_window destructor");

    if ( pattern_pict )
        ReleaseResource ( (Handle)pattern_pict );

    if ( solid_pict )
        ReleaseResource ( (Handle)solid_pict );

    if ( next_button )
        DisposeControl ( next_button );

    if ( cancel_button )
        DisposeControl ( cancel_button );

    dispose_slider( &amb_slider );

    if ( the_palette )
        DisposePalette(the_palette);

    copy_gamma_to_dev( globals->this_component.saved_dev_info );

    do_Show_Menu_Bar ( true );

    DEBUG_PRINT("Left o_black_level_window destructor");
}

// _____

#pragma mark -

// _____

Boolean o_black_level_window::do_Handle_Content_Click ( EventRecord *event )
{
    ControlHandle      control = NULL;
    ControlPartCode    part_code;
    Boolean            click_handled = false;
    Point              where;

    DEBUG_PRINT("Entered o_black_level_window::do_Handle_Content_Click()");

    where = event->where;
    GlobalToLocal ( &where ); // the current port must be correct or GlobalToLocal won't work right

    control = FindControlUnderMouse ( where, window_ref, &part_code );

    if ( control )
    {
        DEBUG_VAR_PRINT("FindControlUnderMouse() found control = %#010x",control);
        DEBUG_EXTRA_VAR_PRINT(", part_code = %d",part_code);

        if ( part_code != kControlNoPart && IsControlVisible(control) && IsControlActive(control) )
        {
            part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
            DEBUG_VAR_PRINT("HandleControlClick() returned part_code = %d",part_code);

            if ( part_code )
            {
                if ( control == next_button )
                {
                    // What color was the user on?
                    switch ( which_color )
                    {
                        case 1:
                        {
                            (globals->this_component).cp_r->black_level = amb_slider.current_value;
                            break;
                        }
                        case 2:
                        {
                            (globals->this_component).cp_g->black_level = amb_slider.current_value;
                            SetControlTitle ( next_button, "\pDone" );
                            break;
                        }
                        case 3:
                        {
                            (globals->this_component).cp_b->black_level = amb_slider.current_value;
                            break;
                        }
                    }
                }

                if ( which_color < 3 )
                {
                    which_color++;
                    do_Init_Black_Level ();
                }
                else
            }
        }
    }
}
```

```
        {
            globals->black_level_window = NULL;
            globals->black_level_complete = true;
            // We should be able to turn off double-buffering and get an update
            // when the window closes, but we'll leave it this way for now.
            ((o_base_window *) (globals->asst_dialog))->do_Force_Update();
            delete this;
        }

        click_handled = true;
    }
    else if ( control == cancel_button )
    {
        globals->black_level_window = NULL;
        globals->black_level_complete = false;
        ((o_base_window *) (globals->asst_dialog))->do_Force_Update();
        delete this;

        click_handled = true;
    }
}

}

}

DEBUG_PRINT("Left o_black_level_window::do_Handle_Content_Click()");

return ( click_handled );
}

//
void o_black_level_window::do_Draw ()
{
    Rect      pict_rect = {0,0,0,0};
    Rect      clip_rect = {0,0,0,0};
    Rect      port_rect = do_Get_Port_Rect();
    RgnHandle  saved_clip_rgn;
    short     left, right, top, bottom, height, width;
    short     x, y, x_steps, y_steps, tile_size;

    DEBUG_PRINT("Entered o_black_level_window::do_Draw()");

    if ( pattern_pict && solid_pict )
    {
        saved_clip_rgn = NewRgn();
        GetClip ( saved_clip_rgn );

//      tile_size = (**pattern_pict).picFrame.right - (**pattern_pict).picFrame.left;
//      tile_size = (**solid_pict).picFrame.right - (**solid_pict).picFrame.left;

        if ( 1 )
        {
            // Unlike the response measurement, we can use the same
            // pattern for all types of displays.

            // So that it looks good, we want an odd number of tiles
            // so that the pattern looks mirrored.

            if ( ( kPreferredPatternWidth / tile_size ) % 2 == 0 )
                width = ( ( kPreferredPatternWidth / tile_size ) - 1 ) * tile_size;
            else
                width = ( kPreferredPatternWidth / tile_size ) * tile_size;

            if ( ( kPreferredPatternHeight / tile_size ) % 2 == 0 )
                height = ( ( kPreferredPatternHeight / tile_size ) - 1 ) * tile_size;
            else
                height = ( kPreferredPatternHeight / tile_size ) * tile_size;

            DEBUG_VAR_PRINT("width: %d",width);
            DEBUG_VAR_PRINT("height: %d",height);

            x_steps = ( width / tile_size );
            y_steps = ( height / tile_size );

            DEBUG_VAR_PRINT("x_steps: %d",x_steps);
            DEBUG_VAR_PRINT("y_steps: %d",y_steps);

            left = ( ( port_rect.right - port_rect.left ) - width ) / 2;
            right = left + width;

            // Place center of pattern 1/3 of the way down the screen
            // since this seems to be the most common focus of attention.
            // On CRTs and projectors, this isn't too important, but on LCDs,
            // the difference in response from top to bottom is really significant.

            top = ( ( port_rect.bottom - port_rect.top ) / 3 ) - ( height / 2 );
            bottom = top + height;
```

```
SetRect ( &clip_rect, left, top, right, bottom );
ClipRect ( &clip_rect );
DEBUG_VAR_PRINT("Clip rect for pattern: %d",clip_rect.left);
DEBUG_EXTRA_VAR_PRINT(" ", %d",clip_rect.top);
DEBUG_EXTRA_VAR_PRINT(" ", %d",clip_rect.right);
DEBUG_EXTRA_VAR_PRINT(" ", %d",clip_rect.bottom);

DEBUG_PRINT("Drawing the pattern picts...");

for ( x = 0; x < x_steps; x++ )
{
    DEBUG_PRINT("(row,col):");

    for ( y = 0; y < y_steps; y++ )
    {
        pict_rect.left = left + ( x * tile_size );
        pict_rect.top = top + ( y * tile_size );
        pict_rect.right = pict_rect.left + tile_size;
        pict_rect.bottom = pict_rect.top + tile_size;

        if ( ( ( x + 2 ) % 2 == 0 && ( y + 2 ) % 2 == 0 ) ||
            ( ( x + 2 ) % 2 != 0 && ( y + 2 ) % 2 != 0 ) )
        {
            if ( ( x == ( x_steps / 2 ) ) && ( y == ( y_steps / 2 ) ) )
            {
                DrawPicture ( solid_pict, &pict_rect );
                DEBUG_EXTRA_PRINT(" (sol)");
            }
            else
            {
                DrawPicture ( pattern_pict, &pict_rect );
                DEBUG_EXTRA_PRINT(" (pat)");
            }
        }
        else
        {
            DEBUG_EXTRA_PRINT(" (spc)");
        }

        DEBUG_EXTRA_VAR_PRINT(",%d",x);
        DEBUG_EXTRA_VAR_PRINT(",%d",y);
    }
}
else
{
    left = ( ( port_rect.right - port_rect.left ) - tile_size ) / 2;
    right = left + tile_size;
    top = ( ( port_rect.bottom - port_rect.top ) / 3 ) - ( tile_size / 2 );
    bottom = top + tile_size;

    SetRect ( &clip_rect, left, top, right, bottom );
    ClipRect ( &clip_rect );

    pict_rect.left = left;
    pict_rect.top = top;
    pict_rect.right = pict_rect.left + tile_size;
    pict_rect.bottom = pict_rect.top + 256;

    DrawPicture ( solid_pict, &pict_rect );
}

SetClip( saved_clip_rgn );
DisposeRgn( saved_clip_rgn );
}

RgnHandle vis_rgn = NewRgn();
UpdateControls ( (WindowPtr)window_ref, do_Get_Visible_Region(vis_rgn) );
DisposeRgn(vis_rgn);

DEBUG_PRINT("Left o_black_level_window::do_Draw()");
}

//
#pragma mark -
//

/* adjust_amb is called a the ambient (or black level) is being dragged */
pascal void o_black_level_window::adjust_amb(ControlHandle this_control,short part_code)
{
    struct calibration_component_info *this_component;
    float value;
    struct f_color temp_color;
    struct components three_colors[3];

    /* pull out the pointer that we need to get to our data */
}
```

```
this_component = (struct calibration_component_info *)GetControlReference(this_control);

/* get the value in floating point. this slider uses a narrow range */
value = slider_to_value(GetControlValue(this_control),this_component->amb_slider);

/* set and draw just the center for black level */
value_to_comp_color(value,&temp_color,this_component);
color_float_to_pixel(&temp_color,&(three_colors[2]),this_component);

value_to_comp_color(value,&temp_color,this_component);
color_float_to_pixel(&temp_color,&(three_colors[1]),this_component);

value_to_comp_color(BBUMP(value),&temp_color,this_component);
color_float_to_pixel(&temp_color,&(three_colors[0]),this_component);

update_colors(this_component->this_dev_info,three_colors,3);
}

//
void o_black_level_window::do_Init_Black_Level ()
{
struct f_color temp_color;
struct components three_colors[3];

switch ( which_color )
{
case 1:
{
(globals->this_component).component_color.red = 1.0;
(globals->this_component).component_color.green = 0.0;
(globals->this_component).component_color.blue = 0.0;
break;
}
case 2:
{
(globals->this_component).component_color.red = 0.0;
(globals->this_component).component_color.green = 1.0;
(globals->this_component).component_color.blue = 0.0;
break;
}
case 3:
{
(globals->this_component).component_color.red = 0.0;
(globals->this_component).component_color.green = 0.0;
(globals->this_component).component_color.blue = 1.0;
break;
}
}

force_slider_position(0.5*AMB_MAX,&amb_slider);

value_to_comp_color(amb_slider.current_value,&temp_color,&(globals->this_component));
color_float_to_pixel(&temp_color,&(three_colors[2]),&(globals->this_component));

value_to_comp_color(amb_slider.current_value,&temp_color,&(globals->this_component));
color_float_to_pixel(&temp_color,&(three_colors[1]),&(globals->this_component));

value_to_comp_color(BBUMP(amb_slider.current_value),&temp_color,&(globals->this_component));
color_float_to_pixel(&temp_color,&(three_colors[0]),&(globals->this_component));

update_colors((globals->this_component).this_dev_info,three_colors,3);
}
```



```
//  
// _____ ©1998-2001 bergdesign inc.  
//  
  
#include "o_response_window.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
o_response_window::o_response_window ( Rect *bounds, Boolean visible, WindowAttributes attributes,  
                                       ThemeBrush brush, WindowRef behind, struct cal_globals *cal_globals )  
    : o_base_window ( bounds, NULL, NULL, visible, attributes, brush, behind )  
{  
    Rect    button_rect, port_rect;  
    int     center = 0;  
    int     error;  
  
    DEBUG_PRINT("Entered o_response_window constructor");  
  
    globals = cal_globals;  
  
    do_Set_Title ( "\pResponse Measurement" );  
    do_Show_Menu_Bar ( false );  
  
    error = linear_gamma_to_dev ( globals->this_component.this_dev_info );  
  
    the_palette = NULL;  
    // initialize_palette ( (WindowPtr>window_ref, &the_palette );  
  
    pattern_pict = NULL;  
    solid_pict = NULL;  
  
    which_color = 1;  
  
    port_rect = do_Get_Port_Rect();  
  
    // vertical slider  
    SetRect ( &(adj_slider.slider_rect), port_rect.right - 24, port_rect.top + 4, port_rect.right - 8,  
port_rect.bottom - 64 );  
    adj_slider.my_window = (WindowPtr>window_ref;  
    adj_slider.live_function = (ProcPtr)(o_response_window::adjust_level);  
    adj_slider.data_pointer = &(globals->this_component);  
    adj_slider.slider_min = 0.0;  
    adj_slider.slider_max = 1.0;  
    initialize_slider(&adj_slider);  
    globals->this_component.adj_slider = &adj_slider;  
    ShowControl(adj_slider.this_control);  
    ActivateControl(adj_slider.this_control);  
  
    // horizontal slider  
    SetRect ( &(pat_slider.slider_rect), port_rect.left + 4, port_rect.bottom - 64, port_rect.right - 32,  
port_rect.bottom - 48 );  
    pat_slider.my_window = (WindowPtr>window_ref;  
    pat_slider.live_function = (ProcPtr)(o_response_window::change_pattern);  
    pat_slider.data_pointer = &(globals->this_component);  
    pat_slider.slider_min = 0.0;  
    pat_slider.slider_max = 1.0;  
    initialize_slider(&pat_slider);  
    globals->this_component.pat_slider = &pat_slider;  
    ShowControl(pat_slider.this_control);  
    ActivateControl(pat_slider.this_control);  
  
    // next button  
    SetRect ( &button_rect, port_rect.right - 122, port_rect.bottom - 32, port_rect.right - 12, port_rect.bottom -  
12 );  
    next_button = NewControl ( window_ref, &button_rect, "\pNext Color", false, 0, 0, 1,  
(short)kControlPushButtonProc, (long)this );  
    ShowControl(next_button);  
  
    // undo button  
    OffsetRect ( &button_rect, -110 - 12, 0 );  
    undo_button = NewControl ( window_ref, &button_rect, "\pUndo", false, 0, 0, 1, (short)kControlPushButtonProc,  
(long)this );  
    ShowControl(undo_button);  
  
    // reset button  
    center = ( port_rect.right - port_rect.left ) / 2;  
    SetRect ( &button_rect, center - 40, port_rect.bottom - 32, center + 40, port_rect.bottom - 12 );  
    reset_button = NewControl ( window_ref, &button_rect, "\pReset", false, 0, 0, 1, (short)kControlPushButtonProc,  
(long)this );  
    ShowControl(reset_button);  
  
    // cancel button  
    SetRect ( &button_rect, port_rect.left + 12, port_rect.bottom - 32, port_rect.left + 92, port_rect.bottom - 12  
);
```

```
cancel_button = NewControl ( window_ref, &button_rect, "\pCancel", false, 0, 0, 1,
(short)kControlPushButtonProc, (long)this );
ShowControl(cancel_button);

// Scale the plot axes
globals->this_component.plot_scale->x_min = 0.0;
globals->this_component.plot_scale->x_max = 1.0;
globals->this_component.plot_scale->y_min = 0.0;
globals->this_component.plot_scale->y_max = 1.0;

// Set the plot size
int plot_width = ( ( port_rect.right - port_rect.left - kPreferredPatternWidth ) / 2 ) - 24 - 12 - 12;
plot_width = MIN(plot_width,256);
int plot_height = plot_width;
globals->this_component.plot_scale->draw_rect.right = port_rect.right - 24 - 12;
globals->this_component.plot_scale->draw_rect.bottom = port_rect.bottom - 64 - 18;
globals->this_component.plot_scale->draw_rect.left = globals->this_component.plot_scale->draw_rect.right -
plot_width - 2;
globals->this_component.plot_scale->draw_rect.top = globals->this_component.plot_scale->draw_rect.bottom -
plot_height - 2;

// Initialize the plot
initialize_scale(globals->this_component.plot_scale);

do_Init_Response ();

if ( globals->display_type == kDisplayTypeCRT )
    pattern_pict = GetPicture ( kCalibrationLines_1 );
else if ( globals->display_type == kDisplayTypeLCD )
    pattern_pict = GetPicture ( kCalibrationPattern_1x1 );
else // kDisplayTypeProjector
    pattern_pict = GetPicture ( kCalibrationPattern_2x2 );

solid_pict = GetPicture ( kCalibrationSolid_008 );

if ( !pattern_pict || !solid_pict )
    do_One_Button_Alert ( kAlertStopAlert, "\pThere was not enough memory to load the calibration picture.",
"\pOK" );
    "\pPlease use Get Info and increase the preferred memory for Monitors & Sound by 25

// do_Force_Update();
do_Force_Draw();

DEBUG_PRINT("Left o_response_window constructor");
}

//
o_response_window::~o_response_window ()
{
    DEBUG_PRINT("Entered o_response_window destructor");

    if ( pattern_pict )
        ReleaseResource ( (Handle)pattern_pict );

    if ( solid_pict )
        ReleaseResource ( (Handle)solid_pict );

    if ( cancel_button )
        DisposeControl ( cancel_button );

    if ( next_button )
        DisposeControl ( next_button );

    if ( reset_button )
        DisposeControl ( reset_button );

    if ( undo_button )
        DisposeControl ( undo_button );

    dispose_slider( &pat_slider );
    dispose_slider( &adj_slider );

    if ( the_palette )
        DisposePalette(the_palette);

    copy_gamma_to_dev( globals->this_component.saved_dev_info );

    do_Show_Menu_Bar ( true );

    DEBUG_PRINT("Left o_response_window destructor");
}

//
#pragma mark -
```

```
//
Boolean o_response_window::do_Handle_Content_Click ( EventRecord *event )
{
ControlHandle          control = NULL;
ControlPartCode        part_code;
Boolean               click_handled = false;
Point                 where;
struct calibration_component_info *this_component;
struct f_color         temp_color;
struct components      three_colors[3];

    DEBUG_PRINT("Entered o_response_window::do_Handle_Content_Click()");

    this_component = &(globals->this_component);

    where = event->where;
    GlobalToLocal ( &where ); // the current port must be correct or GlobalToLocal won't work right

    control = FindControlUnderMouse ( where, window_ref, &part_code );
    if ( control )
    {
        DEBUG_VAR_PRINT("FindControlUnderMouse() found control = %#010x",control);
        DEBUG_EXTRA_VAR_PRINT(", part_code = %d",part_code);

        if ( part_code != kControlNoPart && IsControlVisible(control) && IsControlActive(control) )
        {
            if ( control == cancel_button )
            {
                part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
                if ( part_code )
                {
                    Sint16 answer = do_Two_Button_Alert ( kAlertCautionAlert, "\pAre you sure you want to cancel th
response measurement step?", "\pYour current measurements will be lost.", "\pCancel", "\pContinue" );

                    if( kAlertStdAlertOKButton == answer )
                    {
                        reset_control_points ( this_component->cp_r );
                        reset_control_points ( this_component->cp_g );
                        reset_control_points ( this_component->cp_b );
                        calculate_smoothing ( my_control_points );

                        // Remember - the class is going out of scope and only
                        // this function's local variables are still in scope.
                        globals->response_window = NULL;
                        globals->response_complete = false;
                        // We should be able to turn off double-buffering and get an update
                        // when the window closes, but we'll leave it this way for now.
                        ((o_base_window *) (globals->asst_dialog))->do_Force_Update();
                        delete this;
                    }
                }
            }

            click_handled = true;
        }
        else if ( control == next_button )
        {
            part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
            if ( part_code )
            {
                if( which_color == 2 )
                    SetControlTitle ( next_button, "\pDone" );

                if ( which_color < 3 )
                {
                    calculate_smoothing(this_component->cp_cur);
                    which_color++;
                    do_Init_Response ();
                }
                else
                {
                    // Remember - the class is going out of scope and only
                    // this function's local variables are still in scope.
                    globals->response_window = NULL;
                    globals->response_complete = true;
                    ((o_base_window *) (globals->asst_dialog))->do_Force_Update();
                    delete this;
                }
            }
        }

        click_handled = true;
    }
    else if ( control == reset_button )
    {
        part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
        if ( part_code )
        {
            Sint16 answer = do_Two_Button_Alert ( kAlertCautionAlert, "\pAre you sure you want to reset the
```



```
measurements for the current channel?", NULL, "\pReset", "\pCancel" );

    if( kAlertStdAlertOKButton == answer )
    {
        reset_control_points(this_component->cp_cur);
        calculate_smoothing(this_component->cp_cur);
        make_shifted_control_points(this_component);
        locate_new_test_points(this_component);
        this_component->nearest_point = 1;

        draw_plot(this_component);

value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.a_y),&temp_color,
this_component);
        offset_scale_f_pixel(&temp_color,this_component);
        color_float_to_pixel(&temp_color,&(three_colors[2]),this_component);

value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.b_y),&temp_color,
this_component);
        offset_scale_f_pixel(&temp_color,this_component);
        color_float_to_pixel(&temp_color,&(three_colors[1]),this_component);

value_to_comp_color((this_component->test_points[this_component->nearest_point].y),&temp_color,this_component);
        offset_scale_f_pixel(&temp_color,this_component);
        color_float_to_pixel(&temp_color,&(three_colors[0]),this_component);

        update_colors(this_component->this_dev_info,three_colors,3);

        set_adj_scale(this_component);

        #if PAT_ON_Y
            force_slider_position(get_y_from_x(0.5,this_component->cp_cur),this_component->pat_slid
        #elif PAT_PERCEPT
            force_slider_position(lum_to_percept(0.5),this_component->pat_slider);
        #else
            force_slider_position(0.5,this_component->pat_slider);
        #endif

        ActivateControl(this_component->adj_slider->this_control);
        ActivateControl(this_component->pat_slider->this_control);

        this_component->new_point = 1;

        DeactivateControl(undo_button);
        DeactivateControl(next_button); // make the user calibrate at least one
    }
}

click_handled = true;
}
else if ( control == undo_button )
{
    part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
    if ( part_code )
    {
        do_Undo_Last_Point();
    }

    click_handled = true;
}
else if ( control == pat_slider.this_control )
{
    calculate_smoothing(this_component->cp_cur);
    make_shifted_control_points(this_component);
    locate_new_test_points(this_component);

    this_component->new_point = 1;
    ActivateControl(next_button); // Now it's ok to let them leave

    part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
    DEBUG_VAR_PRINT("HandleControlClick() returned part_code = %d",part_code);

    #if PAT_ON_Y
        force_slider_position(get_y_from_x(this_component->test_points[this_component->nearest_point].x,this_component->
cp_cur),this_component->pat_slider);
    #elif PAT_PERCEPT
        force_slider_position(lum_to_percept(this_component->test_points[this_component->nearest_point].x),this_component->
pat_slider);
    #else
        force_slider_position(this_component->test_points[this_component->nearest_point].x,this_component->pat_slider);
    #endif
}
```

```
        set_adj_scale(this_component);
        click_handled = true;
    }
    else if ( control == adj_slider.this_control )
    {
        DeactivateControl(next_button); // Since they've started to calibrate, don't let them leave
        ActivateControl(undo_button);    // Now you can undo this point

        if ( this_component->new_point ) // Get ready for the change
        {
            this_component->new_cp_index = get_new_cp(this_component);
            this_component->test_points[this_component->nearest_point].point_status = 1;
            this_component->new_point = 0;
        }

        part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
        DEBUG_VAR_PRINT("HandleControlClick() returned part_code = %d",part_code);

        #if PAT_ON_Y

force_slider_position(get_y_from_x(this_component->test_points[this_component->nearest_point].x,this_component->
cp_cur),this_component->pat_slider);
        #endif

        click_handled = true;
    }
}

DEBUG_PRINT("Left o_response_window::do_Handle_Content_Click()");
return ( click_handled );
}

//
Boolean o_response_window::do_Handle_Key_Down( EventRecord *event )
{
    Boolean    key_was_handled = false;
    long       the_key;

    the_key = event->message & charCodeMask;

    if( ( event->modifiers & cmdKey ) && ( the_key == 'z' || the_key == 'Z' ) )
    {
        do_Undo_Last_Point();
        key_was_handled = true;
    }
    else if( ( event->modifiers & cmdKey ) && ( the_key == 'p' || the_key == 'P' ) )
    {
        struct calibration_component_info *this_component = &(globals->this_component);
        write_plot_file( this_component );
        SysBeep(1);
    }
    return( key_was_handled );
}

//
void o_response_window::do_Draw ()
{
    Rect      pict_rect = {0,0,0,0};
    Rect      clip_rect = {0,0,0,0};
    Rect      port_rect = do_Get_Port_Rect();
    RgnHandle saved_clip_rgn = NULL;
    short     left, right, top, middle, bottom, height, width;
    short     x, y, x_steps, y_steps, tile_size;

    DEBUG_PRINT("Entered o_response_window::do_Draw()");

    if ( pattern_pict && solid_pict )
    {
        NormalizeColorAndPen();

        // Save the clip region
        saved_clip_rgn = NewRgn();
        GetClip ( saved_clip_rgn );

        // Calc the width/height of the PICT pattern resource
        tile_size = (**pattern_pict).picFrame.right - (**pattern_pict).picFrame.left;

        // If we're showing the CRT pattern, the number of tiles
        // down the height can be odd, but the number of tiles
        // across the width must be even for things to look right.
        if ( globals->display_type == kDisplayTypeCRT )
        {

```

```
if ( ( kPreferredPatternWidth / tile_size ) % 2 == 0 )
    width = ( kPreferredPatternWidth / tile_size ) * tile_size;
else
    width = ( ( kPreferredPatternWidth / tile_size ) - 1 ) * tile_size;
}
else
{
    width = ( kPreferredPatternWidth / tile_size ) * tile_size;
}

// If we're showing the LCD pattern, the number of tiles
// across the width can be odd, but the number of tiles
// down the height must be even for things to look right.
if ( globals->display_type == kDisplayTypeLCD )
{
    if ( ( kPreferredPatternHeight / tile_size ) % 2 == 0 )
        height = ( kPreferredPatternHeight / tile_size ) * tile_size;
    else
        height = ( ( kPreferredPatternHeight / tile_size ) - 1 ) * tile_size;
}
else
{
    height = ( kPreferredPatternHeight / tile_size ) * tile_size;
}

x_steps = width / tile_size;
y_steps = height / tile_size;

left = ( ( port_rect.right - port_rect.left ) - width ) / 2;
right = left + width;

// Place middle of pattern 1/3 of the way down the screen.
middle = ( port_rect.bottom - port_rect.top ) / 3;
top = middle - ( height / 2 );
bottom = top + height;

// Set a new clip rect so we don't trash the screen
SetRect ( &clip_rect, left, top, right, bottom );
ClipRect ( &clip_rect );

// For an LCD, we put the pattern on top of the solid patch so the
// interface runs horizontally. When it runs vertically, the angle
// of sight makes the top and bottom of the interface look different
// from the middle.
if ( globals->display_type == kDisplayTypeLCD )
{
    y_steps = y_steps / 2;
    for ( x = 0; x < x_steps; x++ )
    {
        for ( y = 0; y < y_steps; y++ )
        {
            pict_rect.left = left + ( x * tile_size );
            pict_rect.top = top + ( y * tile_size );
            pict_rect.right = pict_rect.left + tile_size;
            pict_rect.bottom = pict_rect.top + tile_size;

            DrawPicture ( pattern_pict, &pict_rect );
            DEBUG_VAR_PRINT("Drew the pattern inside %d", pict_rect.left);
            DEBUG_EXTRA_VAR_PRINT(", %d", pict_rect.top);
            DEBUG_EXTRA_VAR_PRINT(", %d", pict_rect.right);
            DEBUG_EXTRA_VAR_PRINT(", %d", pict_rect.bottom);

            pict_rect.top = pict_rect.top + ( height / 2 );
            pict_rect.bottom = pict_rect.top + tile_size;

            DrawPicture ( solid_pict, &pict_rect );
            DEBUG_VAR_PRINT("Drew the solid inside %d", pict_rect.left);
            DEBUG_EXTRA_VAR_PRINT(", %d", pict_rect.top);
            DEBUG_EXTRA_VAR_PRINT(", %d", pict_rect.right);
            DEBUG_EXTRA_VAR_PRINT(", %d", pict_rect.bottom);
        }
    }
}
else if ( globals->display_type == kDisplayTypeProjector )
{
    for ( x = 0; x < x_steps; x++ )
    {
        for ( y = 0; y < y_steps; y++ )
        {
            pict_rect.left = left + ( x * tile_size );
            pict_rect.top = top + ( y * tile_size );
            pict_rect.right = pict_rect.left + tile_size;
            pict_rect.bottom = pict_rect.top + tile_size;

            if ( ( x > 2 ) && ( x < ( x_steps - 3 ) ) && ( y > 2 ) && ( y < ( y_steps - 3 ) ) )
            {
                DrawPicture ( solid_pict, &pict_rect );
            }
        }
    }
}
```

```
        }
        else
        {
            DrawPicture ( pattern_pict, &pict_rect );
        }
    }
}

// For a CRT, we make the interface run vertically so that the line pattern
// smoothly interfaces with the solid patch.
else // kDisplayTypeCRT
{
    x_steps = x_steps / 2;

    for ( y = 0; y < y_steps; y++ )
    {
        for ( x = 0; x < x_steps; x++ )
        {
            pict_rect.left = left + ( x * tile_size );
            pict_rect.top = top + ( y * tile_size );
            pict_rect.right = pict_rect.left + tile_size;
            pict_rect.bottom = pict_rect.top + tile_size;

            DrawPicture ( pattern_pict, &pict_rect );

            pict_rect.left = pict_rect.left + ( width / 2 );
            pict_rect.right = pict_rect.left + tile_size;

            DrawPicture ( solid_pict, &pict_rect );
        }
    }

    // Restore the old clip region
    SetClip( saved_clip_rgn );
    DisposeRgn( saved_clip_rgn );
}

RgnHandle vis_rgn = NewRgn();
UpdateControls ( (WindowPtr)window_ref, do_Get_Visible_Region(vis_rgn) );
DisposeRgn(vis_rgn);

DEBUG_PRINT("Left o_response_window::do_Draw()");
}

//
#pragma mark -
//

/* change_pattern is run as the pattern changing slider is moved */
pascal void o_response_window::change_pattern ( ControlHandle this_control, short part_code )
{
    struct calibration_component_info *this_component;
    float value;
    struct f_color temp_color;
    struct components three_colors[3];

    /* pull out the pointer that we need to get to our data */
    this_component = (struct calibration_component_info *)GetControlReference(this_control);

    /* get the value in floating point from 0.0 - 1.0 */
    value = slider_to_value(GetControlValue(this_control),this_component->pat_slider);

    #if PAT_ON_Y
    /* this expands the scale by using y instead of x */
    value = get_x_from_y(value,this_component->cp_cur);
    #elif PAT_PERCEPT
    /* this remaps from perceptual to luminance */
    value = percept_to_lum(value);
    #endif

    /* locate the closest test point to where the slider is */
    this_component->nearest_point =
    find_nearest_test_point(value,this_component->test_points,this_component->test_point_count);

    /* check to see if it is already calibrated so we can turn off the adjustment slider */
    // if(this_component->test_points[this_component->nearest_point].point_status != 1)
    if(this_component->test_points[this_component->nearest_point].point_status == 0)
    {
        ActivateControl(this_component->adj_slider->this_control);
    }
    else
    {
        DeactivateControl(this_component->adj_slider->this_control);
    }
}
```

```
/* set and draw the gamma test pattern */
value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.a_y),&temp_color,
this_component);
offset_scale_f_pixel(&temp_color,this_component);
color_float_to_pixel(&temp_color,&(three_colors[2]),this_component);

value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.b_y),&temp_color,
this_component);
offset_scale_f_pixel(&temp_color,this_component);
color_float_to_pixel(&temp_color,&(three_colors[1]),this_component);

value_to_comp_color((this_component->test_points[this_component->nearest_point].y),&temp_color,this_component);
offset_scale_f_pixel(&temp_color,this_component);
color_float_to_pixel(&temp_color,&(three_colors[0]),this_component);

update_colors(this_component->this_dev_info,three_colors,3);

/* put up the plot */
draw_plot(this_component);
}

//

/* adjust_level is called as the adjustment slider is dragged */
pascal void o_response_window::adjust_level ( ControlHandle this_control, short part_code )
{
struct calibration_component_info *this_component;
float value;
struct f_color temp_color;
struct components one_color[1];

/* pull out the pointer that we need to get to our data */
this_component = (struct calibration_component_info *)GetControlReference(this_control);

/* get the value in floating point. this slider uses a narrow range */
value = slider_to_value(GetControlValue(this_control),this_component->adj_slider);

/* load that value into both the control and test structures */
this_component->cp_cur->y[this_component->new_cp_index] = value;
this_component->test_points[this_component->nearest_point].y = value;

/* resmooth with the new level */
calculate_smoothing(this_component->cp_cur);

// call bulge thing ( this_component->cp_cur );
// true or false

/* set and draw just the center of the gamma test pattern */
value_to_comp_color(value,&temp_color,this_component);
offset_scale_f_pixel(&temp_color,this_component);
color_float_to_pixel(&temp_color,&(one_color[0]),this_component);

update_colors(this_component->this_dev_info,one_color,1);

/* draw the plot */
draw_plot(this_component);
}

//

void o_response_window::do_Init_Response ()
{
struct f_color temp_color;
struct components three_colors[3];
struct calibration_component_info *this_component;

this_component = &(globals->this_component);

switch ( which_color )
{
case 1:
{
this_component->cp_cur = this_component->cp_r;
this_component->component_color.red = 1.0;
this_component->component_color.green = 0.0;
this_component->component_color.blue = 0.0;
break;
}
case 2:
{
this_component->cp_cur = this_component->cp_g;
this_component->component_color.red = 0.0;
this_component->component_color.green = 1.0;
this_component->component_color.blue = 0.0;
break;
}
}
```

```
        case 3:
        {
            this_component->cp_cur = this_component->cp_b;
            this_component->component_color.red = 0.0;
            this_component->component_color.green = 0.0;
            this_component->component_color.blue = 1.0;
            break;
        }
    }

    /* initialize */
    reset_control_points(this_component->cp_cur);
    calculate_smoothing(this_component->cp_cur);
    make_shifted_control_points(this_component);
    locate_new_test_points(this_component);

    this_component->nearest_point = 1;

    value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.a_y), &temp_color,
    this_component);
    offset_scale_f_pixel(&temp_color, this_component);
    color_float_to_pixel(&temp_color, &(three_colors[2]), this_component);

    value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.b_y), &temp_color,
    this_component);
    offset_scale_f_pixel(&temp_color, this_component);
    color_float_to_pixel(&temp_color, &(three_colors[1]), this_component);

    value_to_comp_color((this_component->test_points[this_component->nearest_point].y), &temp_color, this_component);
    offset_scale_f_pixel(&temp_color, this_component);
    color_float_to_pixel(&temp_color, &(three_colors[0]), this_component);

    update_colors(this_component->this_dev_info, three_colors, 3);

    set_adj_scale(this_component);

    /* draw the plot */
    draw_plot(this_component);

    #if PAT_ON_Y
    force_slider_position(get_y_from_x(0.5, this_component->cp_cur), this_component->pat_slider);
    #elif PAT_PERCEPT
    force_slider_position(lum_to_percept(0.5), this_component->pat_slider);
    #else
    force_slider_position(0.5, this_component->pat_slider);
    #endif

    ActivateControl(this_component->adj_slider->this_control);

    this_component->new_point = 1;

    /* turn off undo control */
    DeactivateControl(undo_button);

    /* we want them to calibrate at least one */
    DeactivateControl(next_button);
}

//
void o_response_window::do_Undo_Last_Point ()
{
    struct calibration_component_info    *this_component;
    struct components                    three_colors[3];
    struct f_color                       temp_color;
    float                                removed_x;

    this_component = &(globals->this_component);

    removed_x = remove_last_cp(this_component);
    calculate_smoothing(this_component->cp_cur);
    make_shifted_control_points(this_component);
    locate_new_test_points(this_component);

    this_component->nearest_point =
    find_nearest_test_point(removed_x, this_component->test_points, this_component->test_point_count);

    draw_plot(this_component);

    value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.a_y), &temp_color,
    this_component);
    offset_scale_f_pixel(&temp_color, this_component);
    color_float_to_pixel(&temp_color, &(three_colors[2]), this_component);
```

```
value_to_comp_color((this_component->test_points[this_component->nearest_point].parents.b_y),&temp_color,
this_component);
    offset_scale_f_pixel(&temp_color,this_component);
    color_float_to_pixel(&temp_color,&(three_colors[1]),this_component);

    value_to_comp_color((this_component->test_points[this_component->nearest_point].y),&temp_color,this_component);
    offset_scale_f_pixel(&temp_color,this_component);
    color_float_to_pixel(&temp_color,&(three_colors[0]),this_component);

    update_colors(this_component->this_dev_info,three_colors,3);

    set_adj_scale(this_component);

    #if PAT_ON_Y
force_slider_position(get_y_from_x((this_component->test_points[this_component->nearest_point].x),this_component->
cp_cur),this_component->pat_slider);
    #elif PAT_PERCEPT
force_slider_position(lum_to_percept((this_component->test_points[this_component->nearest_point].x)),
this_component->pat_slider);
    #else
force_slider_position((this_component->test_points[this_component->nearest_point].y),this_component->pat_slider);
    #endif

    ActivateControl(this_component->adj_slider->this_control);
    ActivateControl(this_component->pat_slider->this_control);

    this_component->new_point = 1;

    /* turn off undo control if you can't undo any more */
    if(this_component->cp_cur->count < 3)
        DeactivateControl(undo_button);
}
```



```

// _____ ©1998-2001 bergdesign inc.
// _____

#include "o_viewer_window.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

// _____

o_viewer_window::o_viewer_window ( Rect *bounds, Boolean visible, WindowAttributes attributes,
                                   ThemeBrush brush, WindowRef behind, struct cal_globals *cal_globals )
    : o_base_window ( bounds, NULL, NULL, visible, attributes, brush, behind )
{
    DEBUG_PRINT("Entered o_viewer_window constructor");

    globals = cal_globals;

    do_Set_Title ( "\pResponse Viewer" );
    do_Show_Menu_Bar ( false );

    int error = linear_gamma_to_dev ( globals->this_component.this_dev_info );

    the_palette = NULL;
// initialize_palette ( (WindowPtr)window_ref, &the_palette );

    which_color = 1;

    Rect port_rect = do_Get_Port_Rect();
    Rect graph_rect = do_Max_Inscribed_Square( port_rect );
// InsetRect( &graph_rect, 48, 48 );
    InsetRect( &graph_rect, 120, 120 );
    OffsetRect( &graph_rect, 0, -12 );

    // Scale the plot axes
    globals->this_component.plot_scale->x_min = 0.0;
    globals->this_component.plot_scale->x_max = 1.0;
    globals->this_component.plot_scale->y_min = 0.0;
    globals->this_component.plot_scale->y_max = 1.0;

    // Set the plot size
    globals->this_component.plot_scale->draw_rect = graph_rect;

    // Initialize the plot
    initialize_scale(globals->this_component.plot_scale);

    // Set the gamma to be used by the plotting routine
    globals->this_component.plot_target_gamma = globals->target_gamma;
    globals->this_component.plot_target_perceptual = globals->target_perceptual;

    globals->this_component.cp_cur = globals->this_component.cp_r;
    calculate_smoothing( globals->this_component.cp_cur );
    make_shifted_control_points( &(globals->this_component) );
    locate_new_test_points( &(globals->this_component) );
    globals->this_component.nearest_point = 1;

    Rect button_rect;
    short button_top = port_rect.bottom - 32;
    short button_bottom = port_rect.bottom - 12;

    // done button
    SetRect ( &button_rect, port_rect.right - 122, button_top, port_rect.right - 12, button_bottom );
    done_button = NewControl ( window_ref, &button_rect, "\pDone", false, 0, 0, 1, (short)kControlPushButtonProc,
(long)this );
    ShowControl(done_button);

    // undo button
// OffsetRect ( &button_rect, -110 - 12, 0 );
    undo_button = NULL;
// undo_button = NewControl ( window_ref, &button_rect, "\pUndo", false, 0, 0, 1, (short)kControlPushButtonProc,
(long)this );
// ShowControl(undo_button);

    // cancel button
// SetRect ( &button_rect, port_rect.left + 12, button_top, port_rect.left + 92, button_bottom );
    cancel_button = NULL;
// cancel_button = NewControl ( window_ref, &button_rect, "\pCancel", false, 0, 0, 1,
(short)kControlPushButtonProc, (long)this );
// ShowControl(cancel_button);

// do_Force_Update();
do_Force_Draw();

    DEBUG_PRINT("Left o_viewer_window constructor");
}

```

```
// _____
o_viewer_window::~o_viewer_window ()
{
    DEBUG_PRINT("Entered o_viewer_window destructor");

    if ( cancel_button )
        DisposeControl ( cancel_button );

    if ( done_button )
        DisposeControl ( done_button );

    if ( undo_button )
        DisposeControl ( undo_button );

    if ( the_palette )
        DisposePalette(the_palette);

    // copy_gamma_to_dev( globals->this_component.saved_dev_info );
    copy_gamma_to_dev( globals->this_component.this_dev_info );

    do_Show_Menu_Bar ( true );

    DEBUG_PRINT("Left o_viewer_window destructor");
}

// _____

#pragma mark -

// _____

Boolean o_viewer_window::do_Handle_Content_Click ( EventRecord *event )
{
    ControlHandle          control = NULL;
    ControlPartCode        part_code;
    Boolean                click_handled = false;
    Point                  where;
    struct calibration_component_info *this_component;

    DEBUG_PRINT("Entered o_viewer_window::do_Handle_Content_Click()");

    this_component = &(globals->this_component);

    where = event->where;
    GlobalToLocal ( &where ); // the current port must be correct or GlobalToLocal won't work right

    control = FindControlUnderMouse ( where, window_ref, &part_code );
    if ( control )
    {
        DEBUG_VAR_PRINT("FindControlUnderMouse() found control = %#010x",control);
        DEBUG_EXTRA_VAR_PRINT("  part_code = %d",part_code);

        if ( part_code != kControlNoPart && IsControlVisible(control) && IsControlActive(control) )
        {
            if ( control == done_button )
            {
                part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
                if ( part_code )
                {
                    // Remember - the class is going out of scope and only
                    // this function's local variables are still in scope.
                    globals->viewer_window = NULL;
                    delete this;
                }

                click_handled = true;
            }
            else if ( control == cancel_button )
            {
                part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
                if ( part_code )
                {
                    // Not handled yet
                }

                click_handled = true;
            }
            else if ( control == undo_button )
            {
                part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
                if ( part_code )
                {
                    // Not handled yet
                }

                click_handled = true;
            }
        }
    }
}
```

```
}
}

DEBUG_PRINT("Left o_viewer_window::do_Handle_Content_Click()");

return ( click_handled );
}

//
Boolean o_viewer_window::do_Handle_Key_Down( EventRecord *event )
{
Boolean    key_was_handled = false;
long       the_key;

    the_key = event->message & charCodeMask;

    if ( ( event->modifiers & cmdKey ) && ( the_key == 'w' || the_key == 'W' ) )
    {
        globals->viewer_window = NULL;
        delete this;
        key_was_handled = true;
    }

    return( key_was_handled );
}

//
void o_viewer_window::do_Draw ()
{
Rect       pict_rect = {0,0,0,0};
Rect       clip_rect = {0,0,0,0};
Rect       port_rect = do_Get_Port_Rect();
RgnHandle  saved_clip_rgn = NULL;

    DEBUG_PRINT("Entered o_viewer_window::do_Draw()");

    // if ( pattern_pict && solid_pict )
    {
        NormalizeColorAndPen();

        // Save the clip region
        saved_clip_rgn = NewRgn();
        GetClip ( saved_clip_rgn );

        ClipRect ( &port_rect );

        draw_all_plots( &(globals->this_component) );

        // Rect text_box = do_Make_Rect(200,200,400,400);
        // do_Draw_Styled_Text( 128, &text_box );
        // unsigned char *the_text = "\pThis is a TETextBox that will automatically wrap and all those other things.";
        // TETextBox( &the_text[1], the_text[0], &text_box, teJustRight );

        // Restore the old clip region
        SetClip( saved_clip_rgn );
        DisposeRgn( saved_clip_rgn );
    }

    RgnHandle vis_rgn = NewRgn();
    UpdateControls ( (WindowPtr)window_ref, do_Get_Visible_Region(vis_rgn) );
    DisposeRgn(vis_rgn);

    DEBUG_PRINT("Left o_viewer_window::do_Draw()");
}
```

```
// _____ ©1998-2001 bergdesign inc.
// _____

#ifndef __o_white_point_window__
#define __o_white_point_window__

#include "o_base_window.h"
// #include "o_cal_slider.h"
#include "o_color_readout.h"

#include "globals.h"
#include "my_menus.h"
#include "cal_math.h"
#include "gamma_utils.h"

// _____

class o_white_point_window : public o_base_window
{
public:
    // constructors & destructors
    o_white_point_window ( Rect *, Boolean, WindowAttributes, ThemeBrush, WindowRef, struct
cal_globals * );
    ~o_white_point_window ();

    Boolean do_Handle_Content_Click ( EventRecord * );
    void do_Draw ();

protected:
    void do_Change_White_Point ( Point );
    void do_Update_Readout();

private:
    // o_cal_slider *h_slider;
    // o_cal_slider *v_slider;
    struct cal_globals *globals;

    ControlHandle done_button;
    ControlHandle native_button;
    ControlHandle revert_button;
    ControlHandle cancel_button;
    ControlHandle gamma_slider;

    float temp_gamma;
    float saved_gamma;

    struct f_color temp_wp;
    struct f_color saved_wp;

    PaletteHandle the_palette;

    PicHandle solid_pict_008;
    PicHandle solid_pict_016;
    PicHandle solid_pict_024;
    PicHandle solid_pict_032;
    PicHandle wp_center_pict;
    PicHandle image_pict;

    float levels[4];
    o_color_readout *color_readout;

    Rect test_image_boundary_rect;
};

#endif /* __o white point window */
```

```
//
//                                     ©1998-2001 bergdesign inc.
//
#include "o_white_point_window.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

//
o_white_point_window::o_white_point_window ( Rect *bounds, Boolean visible, WindowAttributes attributes,
                                             ThemeBrush brush, WindowRef behind, struct cal_globals *cal_globals )
    : o_base_window ( bounds, NULL, NULL, visible, attributes, brush, behind )
{
    int    err = noErr;

    DEBUG_PRINT("Entered o_white_point_window constructor");

    globals = cal_globals;

    // Set gamma values
    saved_gamma = globals->target_gamma;
    temp_gamma = saved_gamma;

    // Set white point values
    saved_wp.red    = globals->this_component.cp_r->white_level;
    saved_wp.green  = globals->this_component.cp_g->white_level;
    saved_wp.blue   = globals->this_component.cp_b->white_level;
    temp_wp = saved_wp;

    // Initialize window settings
    do_Set_Title ( "\pWhite Point Adjustment" );
    do_Show_Menu_Bar ( false );
    err = llinear_gamma_to_dev ( globals->this_component.this_dev_info );

    the_palette = NULL;
    // initialize_palette ( (WindowPtr>window_ref, &the_palette );

    levels[0] = percept_to_lum(0.25);
    levels[1] = percept_to_lum(0.50);
    levels[2] = percept_to_lum(0.75);
    levels[3] = percept_to_lum(1.00);

    solid_pict_008 = GetPicture ( kCalibrationSolid_008 );
    solid_pict_016 = GetPicture ( kCalibrationSolid_016 );
    solid_pict_024 = GetPicture ( kCalibrationSolid_024 );
    solid_pict_032 = GetPicture ( kCalibrationSolid_032 );
    wp_center_pict = GetPicture ( kCalibrationCenter );
    image_pict = GetPicture( 5000 );

    if (
        solid_pict_008 == NULL
        solid_pict_016 == NULL
        solid_pict_024 == NULL
        solid_pict_032 == NULL
        wp_center_pict == NULL
        image_pict == NULL
    )
    {
        do_One_Button_Alert ( kAlertStopAlert,
                             "\pThere was not enough memory to load necessary PICT resources.",
                             "\pThe test pattern may be visually incomplete.",
                             "\pOK" );
    }

    Rect port_rect = do_Get_Port_Rect();
    Rect button_rect;

    test_image_boundary_rect.top    = port_rect.top + 24;           // bottom of color readout
    test_image_boundary_rect.bottom = port_rect.bottom - 68;       // top of gamma slider
    test_image_boundary_rect.left   = port_rect.left;
    test_image_boundary_rect.right  = port_rect.right;

    SetRect ( &button_rect, port_rect.right - 92, port_rect.bottom - 32, port_rect.right - 12, port_rect.bottom - 12 );
    done_button = NewControl ( window_ref, &button_rect, "\pDone", false, 0, 0, 1, (short)kControlPushButtonProc, (long)this );
    ShowControl(done_button);

    short center = ( port_rect.right - port_rect.left ) / 2;
    SetRect ( &button_rect, center - 40, port_rect.bottom - 32, center + 40, port_rect.bottom - 12 );
    native_button = NewControl ( window_ref, &button_rect, "\pNative", false, 0, 0, 1, (short)kControlPushButtonProc, (long)this );
    ShowControl(native_button);

    SetRect ( &button_rect, port_rect.left + 12, port_rect.bottom - 32, port_rect.left + 92, port_rect.bottom - 12 );
    cancel_button = NewControl ( window_ref, &button_rect, "\pCancel", false, 0, 0, 1,

```

```
(short)kControlPushButtonProc, (long)this );
ShowControl(cancel_button);

SetRect ( &button_rect, port_rect.left, port_rect.top, port_rect.right, port_rect.top + 20 );
color_readout = new o_color_readout( window_ref, button_rect, root, false );

SetRect ( &button_rect, port_rect.left + 212, port_rect.bottom - 54, port_rect.right - 112, port_rect.bottom -
38 );
gamma_slider = NewControl( window_ref, &button_rect, "\p", false, 18, 10, 30, (short)(kControlSliderProc +
kControlSliderNonDirectional), (long)this );
// gamma_slider = NewControl( window_ref, &button_rect, "\p", false, 180, 100, 300, (short)(kControlSliderProc +
kControlSliderReverseDirection + kControlSliderHasTickMarks), (long)this );
SetControlValue( gamma_slider, temp_gamma * 10 );
ShowControl(gamma_slider);

float max_wp_saturation;
if( kDisplayTypeProjector == globals->display_type )
    max_wp_saturation = 1.0;
else
    max_wp_saturation = 0.5;

// Initialize the scale of the white point control area
globals->this_component.wp_scale->x_min = -max_wp_saturation;
globals->this_component.wp_scale->x_max = max_wp_saturation;
globals->this_component.wp_scale->y_min = -max_wp_saturation;
globals->this_component.wp_scale->y_max = max_wp_saturation;

int port_width = port_rect.right - port_rect.left;
int port_height = port_rect.bottom - port_rect.top;

if( port_width >= port_height )
{
    globals->this_component.wp_scale->draw_rect.top = port_rect.top;
    globals->this_component.wp_scale->draw_rect.bottom = port_rect.bottom;
    globals->this_component.wp_scale->draw_rect.left = port_rect.left + ( ( port_width - port_height ) / 2 );
    globals->this_component.wp_scale->draw_rect.right = globals->this_component.wp_scale->draw_rect.left +
port_height;
}
else
{
    globals->this_component.wp_scale->draw_rect.top = port_rect.top + ( ( port_height - port_width ) / 2 );
    globals->this_component.wp_scale->draw_rect.bottom = globals->this_component.wp_scale->draw_rect.top +
port_width;
    globals->this_component.wp_scale->draw_rect.left = port_rect.left;
    globals->this_component.wp_scale->draw_rect.right = port_rect.right;
}

// Initialize the plot scale
initialize_scale(globals->this_component.wp_scale);

// do_Force_Update();
do_Force_Draw();

/*
Point port_center;
port_center.h = ( port_rect.right - port_rect.left ) / 2;
port_center.v = ( port_rect.bottom - port_rect.top ) / 2;
// GlobalToLocal ( &port_center ); // the current port must be correct or GlobalToLocal won't work right
do_Change_White_Point( port_center );
*/

// Set the white point in the dev_info
globals->this_component.cp_r->white_level = temp_wp.red;
globals->this_component.cp_g->white_level = temp_wp.green;
globals->this_component.cp_b->white_level = temp_wp.blue;

// Push it to the video card
control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header, false, temp_gamma,
&(globals->this_component) );
copy_gamma_to_dev( globals->this_component.this_dev_info );

do_Update_Readout();

DEBUG_PRINT("Left o_white_point_window constructor");
}

//
o_white_point_window::~o_white_point_window ()
{
    DEBUG_PRINT("Entered o_white_point_window destructor");

    if ( solid_pict_008 )
        ReleaseResource ( (Handle)solid_pict_008 );

    if ( solid_pict_016 )
        ReleaseResource ( (Handle)solid_pict_016 );

    if ( solid_pict_024 )
        ReleaseResource ( (Handle)solid_pict_024 );
}
```

```
if ( solid_pict_032 )
    ReleaseResource ( (Handle)solid_pict_032 );

if ( wp_center_pict )
    ReleaseResource ( (Handle)wp_center_pict );

if ( image_pict )
    ReleaseResource ( (Handle)image_pict );

if ( done_button )
    DisposeControl ( done_button );

if ( native_button )
    DisposeControl ( native_button );

if ( cancel_button )
    DisposeControl ( cancel_button );

if ( gamma_slider )
    DisposeControl ( gamma_slider );

if ( the_palette )
    DisposePalette(the_palette);

if ( color_readout )
    delete color_readout;

do_Show_Menu_Bar ( true );

DEBUG_PRINT("Left o_white_point_window destructor");
}

//
#pragma mark -
//

Boolean o_white_point_window::do_Handle_Content_Click ( EventRecord *event )
{
    Boolean click_handled = false;

    DEBUG_PRINT("Entered o_white_point_window::do_Handle_Content_Click()");

    Point where = event->where;
    GlobalToLocal ( &where ); // the current port must be correct or GlobalToLocal won't work right

    ControlPartCode part_code;
    ControlHandle control = FindControlUnderMouse ( where, window_ref, &part_code );
    if ( control )
    {
        DEBUG_VAR_PRINT("FindControlUnderMouse() found control = %#010x",control);
        DEBUG_EXTRA_VAR_PRINT(", part_code = %d",part_code);

        if ( part_code != kControlNoPart && IsControlVisible(control) && IsControlActive(control) )
        {
            part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
            DEBUG_VAR_PRINT("HandleControlClick() returned part_code = %d",part_code);

            if ( part_code )
            {
                if ( control == native_button )
                {
                    globals->this_component.cp_r->white_level = temp_wp.red      = 1.0;
                    globals->this_component.cp_g->white_level = temp_wp.green    = 1.0;
                    globals->this_component.cp_b->white_level = temp_wp.blue      = 1.0;

                    temp_gamma = 1.8;
                    SetControlValue( gamma_slider, temp_gamma * 10 );

                    control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header, false,
temp_gamma, &(globals->this_component) );
                    copy_gamma_to_dev( globals->this_component.this_dev_info );

                    do_Update_Readout();
                }
                else if ( control == cancel_button )
                {
                    globals->this_component.cp_r->white_level = saved_wp.red;
                    globals->this_component.cp_g->white_level = saved_wp.green;
                    globals->this_component.cp_b->white_level = saved_wp.blue;

                    globals->target_gamma = saved_gamma;

                    globals->white_point_complete = false;

                    control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header, false,
```

```
temp_gamma, &(globals->this_component) );  
    copy_gamma_to_dev( globals->this_component.this_dev_info );  
  
    // Remember - the class is going out of scope and only  
    // this function's local variables are still in scope.  
    globals->white_point_window = NULL;  
    delete this;  
}  
else if ( control == done_button )  
{  
    globals->this_component.cp_r->white_level = temp_wp.red;  
    globals->this_component.cp_g->white_level = temp_wp.green;  
    globals->this_component.cp_b->white_level = temp_wp.blue;  
  
    globals->target_gamma = temp_gamma;  
  
    globals->white_point_complete = true;  
  
    control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header, false,  
temp_gamma, &(globals->this_component) );  
    copy_gamma_to_dev( globals->this_component.this_dev_info );  
  
    // Remember - the class is going out of scope and only  
    // this function's local variables are still in scope.  
    globals->white_point_window = NULL;  
    delete this;  
}  
else if ( control == gamma_slider )  
{  
    short val = GetControlValue( gamma_slider );  
    temp_gamma = (float)val/10.0;  
  
    control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header, false,  
temp_gamma, &(globals->this_component) );  
    copy_gamma_to_dev( globals->this_component.this_dev_info );  
}  
    click_handled = true;  
}  
}  
}  
else  
{  
    // Initialize these so we make one pass through tracking code to start with  
    Point old_where;  
    old_where.h = -1;  
    old_where.v = -1;  
  
#if TARGET_API_MAC_CARBON  
    MouseTrackingResult tracking_result = kMouseTrackingMousePressed;  
#endif  
    do  
    {  
        if ( EqualPt ( where, old_where ) == false )  
        {  
            do_Change_White_Point( where );  
            old_where = where;  
        }  
    }  
  
#if TARGET_API_MAC_CARBON  
    TrackMouseLocation ( NULL, &where, &tracking_result );  
    while ( tracking_result != kMouseTrackingMouseReleased );  
#else  
    GetMouse( &where );  
    SystemTask();  
    while ( StillDown() );  
#endif  
}  
  
DEBUG_PRINT("Left o_white_point_window::do_Handle_Content_Click()");  
  
return ( click_handled );  
}  
  
//  
/*  
void o_white_point_window::do_Draw ()  
{  
    DEBUG_PRINT("Entered o_white_point_window::do_Draw()");  
  
    if ( solid_pict_008 && solid_pict_016 && solid_pict_024 && solid_pict_032 && wp_center_pict )  
    {  
        Rect    pict_rect = {0,0,0,0};  
        Rect    port_rect = do_Get_Port_Rect();  
        short   port_width, port_height;  
        short   pattern_width, pattern_height, pattern_left, pattern_top;
```



```
RgnHandle saved_clip_rgn = NewRgn();
GetClip ( saved_clip_rgn );

port_width = port_rect.right - port_rect.left;
port_height = port_rect.bottom - port_rect.top;

// Place center of pattern 1/3 of the way down the screen
// since this seems to be the most common focus of attention.
// On CRTs and projectors, this isn't too important, but on LCDs,
// the difference in response from top to bottom is really significant.
pattern_height = 128;
pattern_width = pattern_height * 4;
pattern_left = port_rect.left + ( port_width / 2 ) - ( pattern_width / 2 );
pattern_top = ( port_height / 3 ) - ( pattern_height / 2 );

SetRect ( &pict_rect, pattern_left, pattern_top, pattern_left + pattern_height, pattern_top + pattern_height );
};
ClipRect ( &pict_rect );
DrawPicture ( solid_pict_008, &pict_rect );

OffsetRect( &pict_rect, pattern_height, 0 );
ClipRect ( &pict_rect );
DrawPicture ( solid_pict_016, &pict_rect );

OffsetRect( &pict_rect, pattern_height, 0 );
ClipRect ( &pict_rect );
DrawPicture ( solid_pict_024, &pict_rect );

OffsetRect( &pict_rect, pattern_height, 0 );
ClipRect ( &pict_rect );
DrawPicture ( solid_pict_032, &pict_rect );

SetRect ( &pict_rect, ( port_width / 2 ) - 16, ( port_height / 2 ) - 16, ( port_width / 2 ) + 16, (
port_height / 2 ) + 16 );
ClipRect ( &pict_rect );
DrawPicture ( wp_center_pict, &pict_rect );

SetClip( saved_clip_rgn );
DisposeRgn( saved_clip_rgn );
}

RgnHandle vis_rgn = NewRgn();
UpdateControls ( (WindowPtr)window_ref, do_Get_Visible_Region(vis_rgn) );
DisposeRgn(vis_rgn);

DEBUG_PRINT("Left o_white_point_window::do_Draw()");
}
*/
//
void o_white_point_window::do_Draw ()
{
    Rect port_rect = do_Get_Port_Rect();

    if ( image_pict )
    {
        Rect pict_rect = {0,0,680,900}; // t,l,b,r

        RgnHandle saved_clip_rgn = NewRgn();
        GetClip ( saved_clip_rgn );

        pict_rect = do_Max_Inscribed_Rect( pict_rect, test_image_boundary_rect );
        ClipRect ( &pict_rect );
        DrawPicture ( image_pict, &pict_rect );

        Rect grad_rect = do_Make_Rect( pict_rect.left, pict_rect.top, pict_rect.left + 30, pict_rect.bottom );
        RGBColor start_color = do_Make_RGBColor( 65535, 65535, 65535 );
        RGBColor end_color = do_Make_RGBColor( 0, 0, 0 );
        do_Draw_Gradation( grad_rect, start_color, end_color, 20 );

        grad_rect = do_Make_Rect( pict_rect.right - 10, pict_rect.top, pict_rect.right, pict_rect.bottom );
        start_color = do_Make_RGBColor( 0, 0, 65535 );
        end_color = do_Make_RGBColor( 0, 0, 0 );
        do_Draw_Gradation( grad_rect, start_color, end_color, 20 );

        OffsetRect( &grad_rect, -10, 0 );
        start_color = do_Make_RGBColor( 0, 65535, 0 );
        end_color = do_Make_RGBColor( 0, 0, 0 );
        do_Draw_Gradation( grad_rect, start_color, end_color, 20 );

        OffsetRect( &grad_rect, -10, 0 );
        start_color = do_Make_RGBColor( 65535, 0, 0 );
        end_color = do_Make_RGBColor( 0, 0, 0 );
        do_Draw_Gradation( grad_rect, start_color, end_color, 20 );

        SetClip( saved_clip_rgn );
    }
}
```

```
        DisposeRgn( saved_clip_rgn );
    }

    ForeColor(whiteColor);
    MoveTo(port_rect.left + 112, port_rect.bottom - 42 );
    DrawString("\pImage Gamma");

    RgnHandle vis_rgn = NewRgn();
    UpdateControls ( (WindowPtr>window_ref, do_Get_Visible_Region(vis_rgn) );
    DisposeRgn(vis_rgn);
}

//
void o_white_point_window::do_Change_White_Point ( Point where )
{
    //struct components      colors[4];
    struct coordinates      these_coordinates;

    /* covert it to user coords */
    these_coordinates.sx = where.h;
    these_coordinates.sy = where.v;

    /* next convert mouse location to x and y */
    to_value( &these_coordinates, globals->this_component.wp_scale );
    xy_to_color( (double)these_coordinates.fx, (double)these_coordinates.fy, &temp_wp );
}

/*
    // This updates several table indices
    // The FULL_X_TO_Y_PTR NWP macro keeps us from having to save and restore the white point info.
    colors[0].red   = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.red *
levels[0],globals->this_component.cp_r));
    colors[0].green = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.green *
levels[0],globals->this_component.cp_g));
    colors[0].blue  = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.blue *
levels[0],globals->this_component.cp_b));

    colors[1].red   = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.red *
levels[1],globals->this_component.cp_r));
    colors[1].green = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.green *
levels[1],globals->this_component.cp_g));
    colors[1].blue  = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.blue *
levels[1],globals->this_component.cp_b));

    colors[2].red   = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.red *
levels[2],globals->this_component.cp_r));
    colors[2].green = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.green *
levels[2],globals->this_component.cp_g));
    colors[2].blue  = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.blue *
levels[2],globals->this_component.cp_b));

    colors[3].red   = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.red *
levels[3],globals->this_component.cp_r));
    colors[3].green = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.green *
levels[3],globals->this_component.cp_g));
    colors[3].blue  = (unsigned char)(0.5 + 255.0 * FULL_X_TO_Y_PTR_NWP(temp_wp.blue *
levels[3],globals->this_component.cp_b));

    update_colors(globals->this_component.this_dev_info,colors,4);
*/

    // need to undo these
    globals->this_component.cp_r->white_level = temp_wp.red;
    globals->this_component.cp_g->white_level = temp_wp.green;
    globals->this_component.cp_b->white_level = temp_wp.blue;

    control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header, false, temp_gamma,
&(globals->this_component) );
    copy_gamma_to_dev( globals->this_component.this_dev_info );

    do_Update_Readout();
}

//
// Updates the on-screen display

void o_white_point_window::do_Update_Readout()
{
    RGBCharColor    char_color;

    char_color.alpha    = 0;
    char_color.red      = temp_wp.red * 255;
    char_color.green     = temp_wp.green * 255;
    char_color.blue      = temp_wp.blue * 255;

    color_readout->do_Set_Color_Values ( char_color );
}
```

```
//  
//  
//  
#ifndef __o_color_readout__  
#define __o_color_readout__  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#ifdef TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <MacTypes.h>  
#include <Appearance.h>  
#include <Sound.h>  
#include <Quickdraw.h>  
#include <Fonts.h>  
#endif  
#endif  
  
#include "my_quickdraw.h"  
  
typedef struct o_color_readout_event  
{  
    SInt16                current_tab;  
    RGBColor              bkgnd_color;  
}  
ColorReadoutEventRecord;  
  
class o_color_readout  
{  
private:  
  
protected:  
  
    ControlHandle          user_pane_cntl_hndl;  
    ControlUserPaneDrawUPP readout_draw_proc;  
    ControlUserPaneHitTestUPP readout_hit_test_proc;  
    GWorldPtr              gworld_ptr;  
  
    static void            do_Readout_Draw_Proc ( ControlHandle, SInt16 );  
    static ControlPartCode do_Readout_Hit_Test_Proc ( ControlHandle, Point );  
  
public:  
  
    SInt16                alpha;  
    SInt16                red;  
    SInt16                green;  
    SInt16                blue;  
  
    Boolean               use_appearance;  
    RGBColor              fore_color;  
    RGBColor              fore_inactive_color;  
    RGBColor              back_color;  
  
    // constructors & destructor  
    o_color_readout ();  
    o_color_readout ( WindowPtr, Rect, ControlHandle, Boolean );  
    ~o_color_readout ();  
    void do_Init_Vars ();  
    void do_Create ( WindowPtr, Rect, ControlHandle, Boolean );  
  
    // basic functionality  
    void do_Show ( Boolean );  
    void do_Activate ( Boolean );  
    void do_Resize ( Rect );  
    Boolean do_Handle_Click ( EventRecord *, ColorReadoutEventRecord * );  
    void do_Update_Display ();  
  
    // accessors  
    void do_Set_Color_Values ( RGBCharColor );  
};  
  
#endif /* __o_color_readout__ */
```



```
{
    EmbedControl ( user_pane_cntl_hndl, embedder );

//    readout_draw_proc = NewControlUserPaneDrawProc ( o_color_readout::do_Readout_Draw_Proc );
    readout_draw_proc = NewControlUserPaneDrawUPP (
(ControlUserPaneDrawProcPtr)o_color_readout::do_Readout_Draw_Proc );

//    readout_hit_test_proc = NewControlUserPaneHitTestProc ( o_color_readout::do_Readout_Hit_Test_Proc );
    readout_hit_test_proc = NewControlUserPaneHitTestUPP (
(ControlUserPaneHitTestProcPtr)o_color_readout::do_Readout_Hit_Test_Proc );

    SetControlData ( user_pane_cntl_hndl, 0, kControlUserPaneDrawProcTag, sizeof( ControlUserPaneDrawUPP ),
(Ptr)&readout_draw_proc );
    SetControlData ( user_pane_cntl_hndl, 0, kControlUserPaneHitTestProcTag, sizeof( ControlUserPaneHitTestUPP
), (Ptr)&readout_hit_test_proc );

    ShowControl ( user_pane_cntl_hndl );
}

use_appearance = use_app_mgr;

do_Update_Display ();

DEBUG_PRINT ("Created the color readout");
}

//
void o_color_readout::do_Update_Display ()
{
    DrawOneControl ( user_pane_cntl_hndl );
}

//
void o_color_readout::do_Set_Color_Values ( RGBCharColor char_color )
{
    alpha   = char_color.alpha;
    red     = char_color.red;
    green   = char_color.green;
    blue    = char_color.blue;

    do_Update_Display ();
}

//
Boolean o_color_readout::do_Handle_Click ( EventRecord * eventPtr, ColorReadoutEventRecord *
channel_controller_event )
{
Boolean          hit = false;
Point            where;

    where = (*eventPtr).where;
    GlobalToLocal ( &where );

    if ( TestControl ( user_pane_cntl_hndl, where ) != 0 )
    {
        do_Update_Display ();
        hit = true;
    }
    else
    {
        hit = false;
    }

    return hit;
}

//
void o_color_readout::do_Activate ( Boolean activate )
{
    if ( activate )
    {
        ActivateControl ( user_pane_cntl_hndl );
    }
    else
    {
        DeactivateControl ( user_pane_cntl_hndl );
    }
}

//
void o_color_readout::do_Show ( Boolean show )
{
    if ( show )
```

```
{
    ShowControl ( user_pane_cntl_hndl );
}
else
{
    HideControl ( user_pane_cntl_hndl );
}
}

//
void o_color_readout::do_Resize ( Rect control_bounds )
{
    SizeControl ( user_pane_cntl_hndl, ( control_bounds.right - control_bounds.left ), ( control_bounds.bottom -
control_bounds.top ) );

    UpdateGWorld ( &gworld_ptr, 32, &control_bounds, NULL, NULL, NULL );

    DrawOneControl ( user_pane_cntl_hndl );
}

//
void o_color_readout::do_Readout_Draw_Proc ( ControlHandle control, Sint16 part )
{
    RgnHandle          saved_clip_rgn = NULL;
    CGrafPtr           port;
    GDHandle           gdh;
    //extern QDGlobals   qd;
    WindowPtr          window_ptr;
    Rect               control_bounds, gworld_bounds, graph_frame_rect;
    o_color_readout    *class_ptr;
    FontInfo           grafport_font_info;
    int                 baseline_x, baseline_y, font_height, readout_string_width;
    int                 bar_graph_height, bar_graph_left, bar_graph_top, bar_graph_right, bar_graph_bot;
    char                alpha_string[6], red_string[6], green_string[6], blue_string[6];
    //ColorPenState      state, window_state;
    ThemeDrawingState   state, window_state;

    class_ptr = (o_color_readout *)GetControlReference( control );
    // window_ptr = (**control).ctrlOwner;
    window_ptr = GetControlOwner( control );

    if ( IsWindowCollapsed ( window_ptr ) )
        return;

    if ( !class_ptr->gworld_ptr )
        return;

    GetGWorld ( &port, &gdh );
    // GetColorAndPenState( &state );
    GetThemeDrawingState( &state );

    if ( LockPixels ( GetGWorldPixMap ( class_ptr->gworld_ptr ) ) )
    {
        // control_bounds = (**control).ctrlRect;
        GetControlBounds( control, &control_bounds );
        // gworld_bounds = (class_ptr->gworld_ptr)->portRect;
        GetPortBounds( class_ptr->gworld_ptr, &gworld_bounds );

        // switch to the offscreen gworld
        SetGWorld ( class_ptr->gworld_ptr, NULL );
        NormalizeColorAndPen();

        // Set up and measure the size of the text so we can place the labels
        font_height = 9;

        TextFont ( applFont );
        TextFace ( normal );
        TextSize ( font_height );

        GetFontInfo ( &grafport_font_info );

        baseline_y = gworld_bounds.top + ( ( gworld_bounds.bottom - gworld_bounds.top - grafport_font_info.ascent -
grafport_font_info.descent ) / 2 ) + grafport_font_info.ascent;
        baseline_x = gworld_bounds.left + ( ( gworld_bounds.bottom - gworld_bounds.top - grafport_font_info.ascent
grafport_font_info.descent ) / 2 );

        sprintf ( alpha_string, "M:888" );
        do_c2p_str ( alpha_string );
        readout_string_width = StringWidth ( (ConstStr255Param)alpha_string ) + ( StringWidth (
(ConstStr255Param)alpha_string ) / 5 ); // 5 characters in string

        // Set up the metrics for the graph
        bar_graph_height = 14;
        bar_graph_left = baseline_x + ( readout_string_width * 4 );
        bar_graph_top = gworld_bounds.top + ( ( gworld_bounds.bottom - gworld_bounds.top - bar_graph_height ) / 2 )
        bar_graph_right = bar_graph_left + 256;
```

```
bar_graph_bottom = bar_graph_top + bar_graph_height;

sprintf ( alpha_string, "A:%d", class_ptr->alpha );
do_c2p_str ( alpha_string );

sprintf ( red_string, "R:%d", class_ptr->red );
do_c2p_str ( red_string );

sprintf ( green_string, "G:%d", class_ptr->green );
do_c2p_str ( green_string );

sprintf ( blue_string, "B:%d", class_ptr->blue );
do_c2p_str ( blue_string );

// We need to set the foreground color to an appropriate color for the theme.
if ( IsControlActive (control) )
{
    if( class_ptr->use_appearance == true )
    {
        DrawThemePlacard ( &gworld_bounds, kThemeStateActive );
        SetThemeTextColor ( kThemeActiveDialogTextColor, 32, true );
    }
    else
    {
        RGBBackColor ( &(class_ptr->back_color) );
        EraseRect ( &gworld_bounds );
        RGBForeColor ( &(class_ptr->fore_color) );
    }
}
else
{
    if( class_ptr->use_appearance == true )
    {
        DrawThemePlacard ( &gworld_bounds, kThemeStateDisabled );
        SetThemeTextColor ( kThemeInactiveDialogTextColor, 32, true );
    }
    else
    {
        RGBBackColor ( &(class_ptr->back_color) );
        EraseRect ( &gworld_bounds );
        RGBForeColor ( &(class_ptr->fore_inactive_color) );
    }
}

MoveTo ( baseline_x, baseline_y );
DrawString ( (ConstStr255Param)alpha_string );
MoveTo ( baseline_x + readout_string_width, baseline_y );
DrawString ( (ConstStr255Param)red_string );
MoveTo ( baseline_x + (readout_string_width * 2), baseline_y );
DrawString ( (ConstStr255Param)green_string );
MoveTo ( baseline_x + (readout_string_width * 3), baseline_y );
DrawString ( (ConstStr255Param)blue_string );

// should fill in the graph background with a neutral gray
// so dark themes don't screw things up

PenSize ( 1, 1 );
MoveTo ( bar_graph_left + 64, bar_graph_top );
LineTo ( bar_graph_left + 64, bar_graph_bottom - 1 ); // "-1" accounts for pen size
// LineTo ( bar_graph_left + 64, bar_graph_bottom - (*(GrafPtr>window_ptr).pnSize.v ); // can do it this way
too

MoveTo ( bar_graph_left + 128, bar_graph_top );
LineTo ( bar_graph_left + 128, bar_graph_bottom - 1 );
MoveTo ( bar_graph_left + 192, bar_graph_top );
LineTo ( bar_graph_left + 192, bar_graph_bottom - 1 );

PenSize ( 1, 3 );

RGBForeColor ( &RGB_BLACK );
MoveTo ( bar_graph_left, bar_graph_top + 1 );
LineTo ( bar_graph_left + class_ptr->alpha, bar_graph_top + 1 ); // "+1" moves down below frame

RGBForeColor ( &RGB_RED );
MoveTo ( bar_graph_left, bar_graph_top + 4 );
LineTo ( bar_graph_left + class_ptr->red, bar_graph_top + 4 ); // 3 pixel increment accounts for pen size

RGBForeColor ( &RGB_GREEN );
MoveTo ( bar_graph_left, bar_graph_top + 7 );
LineTo ( bar_graph_left + class_ptr->green, bar_graph_top + 7 );

RGBForeColor ( &RGB_BLUE );
MoveTo ( bar_graph_left, bar_graph_top + 10 );
LineTo ( bar_graph_left + class_ptr->blue, bar_graph_top + 10 );

PenSize ( 1, 1 );

// Since we changed the pen color several times, we need to set it back to
// the appropriate theme color.
```

```
if ( IsControlActive (control) )
{
    if( class_ptr->use_appearance == true )
    {
        SetThemeTextColor ( kThemeActiveDialogTextColor, 32, true );
    }
    else
    {
        RGBForeColor ( &(class_ptr->fore_color) );
    }
}
else
{
    if( class_ptr->use_appearance == true )
    {
        SetThemeTextColor ( kThemeInactiveDialogTextColor, 32, true );
    }
    else
    {
        RGBForeColor ( &(class_ptr->fore_inactive_color) );
    }
}

SetRect ( &graph_frame_rect, bar_graph_left, bar_graph_top, bar_graph_right, bar_graph_bottom );
FrameRect ( &graph_frame_rect );

// switch to the onscreen window
// SetGWorld ( (CGrafPtr)window_ptr, NULL );
// SetPortWindowPort(window_ptr);

// GetColorAndPenState( &window_state );
// GetThemeDrawingState( &window_state );

NormalizeColorAndPen();

saved_clip_rgn = NewRgn ();
GetClip ( saved_clip_rgn );
ClipRect ( &control_bounds );

CopyBits ( GetPortBitMapForCopyBits( (GrafPtr)(class_ptr->gworld_ptr) ),
            GetPortBitMapForCopyBits( GetWindowPort(window_ptr) ),
            &(gworld_bounds),
            &(control_bounds),
            srcCopy,
            NULL );

/*
CopyBits ( &( *((GrafPtr)(class_ptr->gworld_ptr)) ).portBits ),
            &( (*window_ptr).portBits ),
            &(gworld_bounds),
            &(control_bounds),
            srcCopy,
            NULL );
*/

SetClip( saved_clip_rgn );
DisposeRgn( saved_clip_rgn );

// SetColorAndPenState( &window_state );
// SetThemeDrawingState( window_state, true );

UnlockPixels ( GetGWorldPixMap ( class_ptr->gworld_ptr ) );
}

SetGWorld ( port, gdh );
// SetColorAndPenState( &state );
// SetThemeDrawingState( state, true );
}

//
ControlPartCode o_color_readout::do_Readout_Hit_Test_Proc ( ControlHandle control, Point where )
{
    Rect bounds;
    ControlPartCode part_code;

    // bounds = (**control).controlRect;
    // GetControlBounds( control, &bounds );
    // InsetRect ( &bounds, 2, 2 );

    if ( PtInRect ( where, &bounds ) )
        part_code = 1;
    else
        part_code = 0;

    return part_code;
}
```

```
//  
//  
//
```

```
#ifndef __o_intro_pane_  
#define __o_intro_pane__
```

```
#include "o_base_asst_pane.h"  
#include "o_help_pane.h"
```

```
class o_intro_pane : public o_base_asst_pane  
{  
    public:  
        o_intro_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );  
        ~o_intro_pane ();  
  
        void do_Item_Hit ( short );  
  
    protected:  
  
    private:  
};  
  
#endif /* __o_intro_pane__ */
```

```
//  
// _____  
// _____  
// _____  
  
#include "o_intro_pane.h"  
  
enum  
{  
    kIntroPaneDITL          = 3500,  
    kIntroPaneAppendMode    = -1  
};  
  
enum  
{  
    kIntroPanePicture        = 1,  
    kIntroStaticText1        = 2,  
    kIntroStaticText2        = 3,  
    kIntroStaticText3        = 4,  
    kExpertModeCheckbox      = 5  
};  
  
// _____  
  
o_intro_pane::o_intro_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kIntroPaneDITL, kIntroPaneAppendMode, cal_globals )  
{  
    do_Activate_DItem( window_ref, kExpertModeCheckbox + num_orig_items, false );  
    do_Set_Value_Of_DItem_As_Boolean ( window_ref, kExpertModeCheckbox + num_orig_items, globals->expert_mode );  
}  
  
// _____  
  
o_intro_pane::~o_intro_pane ()  
{  
}  
  
// _____  
  
void  
o_intro_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kExpertModeCheckbox:  
        {  
            globals->expert_mode = !globals->expert_mode;  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kExpertModeCheckbox + num_orig_items, globals->expert_mo  
        );  
            break;  
        }  
        default:  
        {  
            break;  
        }  
    }  
}
```



```
//  
// _____  
// _____  
// _____  
  
#include "o_new_or_adjust_pane.h"  
#include "my_controls.h"  
  
enum  
{  
    kNewOrAdjustPaneDITL          = 3800,  
    kNewOrAdjustPaneAppendMode    = -1  
};  
  
enum  
{  
    kNewOrAdjustPaneStaticText1    = 1,  
    kNewOrAdjustPaneStaticText2    = 2,  
    kNewOrAdjustPaneStaticText3    = 3,  
    kNewOrAdjustPaneProfileMenu    = 4,  
    kNewOrAdjustPaneText01         = 6,  
    kNewOrAdjustPaneText02         = 8,  
    kNewOrAdjustPaneText03         = 10,  
    kNewOrAdjustPaneText04         = 12,  
    kNewOrAdjustPaneText05         = 14,  
    kNewOrAdjustPaneText06         = 16,  
    kNewOrAdjustPaneText07         = 18,  
    kNewOrAdjustPaneText08         = 20,  
    kNewOrAdjustPaneText09         = 22  
};  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
o_new_or_adjust_pane::o_new_or_adjust_pane ( DialogRef dialog, short items, struct cal_globals *globals )  
    : o_base_asst_pane ( dialog, items, kNewOrAdjustPaneDITL, kNewOrAdjustPaneAppendMode, globals )  
{  
    // Set the pop-up menu to the right item  
    if( globals->chosen_profile_loc.locType == cmNoProfileBase )  
    {  
        do_Set_Value_Of_DItem ( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items, kProfileMenuNewProfileIte  
    );  
    }  
    else  
    {  
        do_Set_Value_Of_DItem ( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items,  
globals->chosen_profile_index + kProfileMenuFirstProfileItem );  
        do_Restore_Info_Fields();  
    }  
  
    do_Draw_One_Control_As_DItem( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items );  
}  
  
// _____  
  
o_new_or_adjust_pane::~o_new_or_adjust_pane ()  
{  
}  
  
// _____  
  
void  
o_new_or_adjust_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kNewOrAdjustPaneProfileMenu:  
        {  
            short menu_item_hit = 0;  
            do_Get_Value_Of_DItem ( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items, &menu_item_hit );  
            int profile_hit = do_Menu_Item_To_Profile_Index( menu_item_hit );  
  
            // We only need to do something if the user's new choice is different from the previous choice.  
            if( profile_hit != globals->chosen_profile_index )  
            {  
                // If the user chose to start a new profile...  
                if( profile_hit == -1 )  
                {  
                    // Did the user make any changes to the existing profile?  
  
                    // We reset the measurement values from the existing profile.  
                    // We won't bother annoying the user because no data is lost  
                    // when switching from an existing profile to a new profile.  
                    // The user can always choose the existing profile again.  
                    (globals->chosen_profile_loc).locType = cmNoProfileBase;  
                }  
            }  
        }  
    }  
}
```

```
globals->chosen_profile_index = -1;
globals->black_level_complete = false;
globals->response_complete = false;

do_Clear_Info_Fields();
}
else // User chose to use an existing profile
{
    // Did the user start to make any measurements or make any changes?
    SInt16 answer = kAlertStdAlertOKButton;

    // Was it previously a new profile?
    if( globals->chosen_profile_index == -1 )
    {
        // Did the user begin measuring?
        if( globals->black_level_complete )
        {
            // If so, we ask them if they want to overwrite them.
            answer = do_Two_Button_Alert ( kAlertCautionAlert,
                "\pChanging to an existing profile will invalidate your
measurements. Do you want to continue?",
                "\pThis action cannot be undone.",
                "\pContinue",
                "\pCancel" );
        }
    }
    else // It was an existing profile.
    {
        // Did the user make any changes?.
        if( globals->made_changes )
        {
        }
    }

    // If they don't care, we do it.
    if( answer == kAlertStdAlertOKButton )
    {
        OSStatus err = do_Get_Profile_Custom_Measurements_Tag(
&(globals->profile_loc_array[profile_hit]) );
        if( err == noErr )
        {
            globals->chosen_profile_loc = globals->profile_loc_array[profile_hit];
            globals->chosen_profile_index = profile_hit;
        }
        else
        {
            do_Set_Value_Of_DItem ( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items,
kProfileMenuNewProfileItem );
        }
        // Otherwise, we put things back like they were.
        else // kAlertStdAlertCancelButton
        {
            do_Set_Value_Of_DItem ( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items,
do_Profile_Index_To_Menu_Item( globals->chosen_profile_index ) );
            do_Draw_One_Control_As_DItem( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items );
        }
    }
}

break;
}
default:
{
    break;
}
}

//
CMError o_new_or_adjust_pane::do_Get_Profile_Custom_Measurements_Tag ( CMPProfileLocation *prof_loc_ptr )
{
    CMError      err = noErr;
    Boolean      found_it = false;
    CMPProfileRef prof_ref = NULL;
    UInt32      elem_size = 0;
    char        *elem_data = NULL;

    CMOpenProfile( &prof_ref, prof_loc_ptr );
    if( prof_ref != NULL )
    {
        err = CMPProfileElementExists ( prof_ref, kIccPrivateTag, &found_it );
        if( noErr == err && found_it == true )
        {
            // get the element size
            err = CMGetProfileElement( prof_ref, kIccPrivateTag, &elem_size, NULL );
            if( noErr == err )

```

```
{
    elem_data = (char *)calloc( 1, elem_size );
    if( elem_data != NULL )
    {
        err = CMGetProfileElement( prof_ref, kIccPrivateTag, &elem_size, elem_data );
        if( noErr == err )
        {
            char    *data_start = elem_data;
            UInt32  data_size = elem_size;

            // We need to eat the first tag. It's the kIccPrivateTag
            // used by the profile, not one of our subtags.
            data_start += sizeof(UInt32);
            data_size -= sizeof(UInt32);

            // Thanks to the very early versions of SuperCal, we need to check and see
            // if the custom data is the new format. This is probably not foolproof
            // since the old data started with a float, but it's damn close.

            // We made a mistake in a late alpha build and didn't include the reserved field
            // for our private data tag. After inserting the required 4 bytes and setting them to
            0x00000000,

            // we now need to jump over this field, but we still need to support the few profiles
            // built with the reserved field missing. To do so, we try and validate these 4 bytes.
            // If it works, we have an early alpha build profile. Otherwise, we skip the 4 bytes
            // and try again. After that, something is definitely wrong, so we bail.
            if( !do_Validate_Tag( data_start ) )
            {
                data_start += sizeof(UInt32);
                data_size -= sizeof(UInt32);
            }

            if( do_Validate_Tag( data_start ) )
            {
                char    *data = data_start;
                UInt32  length = data_size;

                // Find the 'chan' tag
                err = do_Find_Tag( &data, &length, 'chan' );
                if( err == noErr )
                {
                    // This tells us how many control point structures to read in from the 'rimd' tag.
                    globals->number_of_channels = *((UInt32 *)data);
                    if( globals->number_of_channels == 1 )
                        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText01 +
num_orig_items, "\pOff", false );
                    else
                        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText01 +
num_orig_items, "\pOn", false );

                    // reset the pointer
                    data = data_start;
                    length = data_size;

                    // Find the 'rimd' tag
                    err = do_Find_Tag( &data, &length, 'rimd' );
                    if( err == noErr )
                    {
                        // The first 4 bytes of the 'rimd' tag is the structure version
                        UInt32 version = *((UInt32 *)data);
                        data += sizeof(UInt32);

                        if( version == 0x00000100 )
                        {
                            if( globals->number_of_channels == 3 )
                            {
                                stream_in_control_point_info( globals->this_component.cp_r, data );
                                data += get_control_point_info_size( globals->this_component.cp_r->count );
                                stream_in_control_point_info( globals->this_component.cp_g, data );
                                data += get_control_point_info_size( globals->this_component.cp_g->count );
                                stream_in_control_point_info( globals->this_component.cp_b, data );
                                data += get_control_point_info_size( globals->this_component.cp_b->count );

                                calculate_smoothing( globals->this_component.cp_r );
                                calculate_smoothing( globals->this_component.cp_g );
                                calculate_smoothing( globals->this_component.cp_b );
                            }

                            // initialize other flags
                            globals->black_level_complete = true;
                            globals->response_complete = true;
                            // white_point ???
                        }
                    }
                    else
                    {
                        do_One_Button_Alert ( kAlertNoteAlert, "\pThe selected profile is not compa
```



```
with this version of SuperCal.", NULL, "\pOK" );
    do_Set_Value_Of_DItem ( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_
kProfileMenuNewProfileItem );
    do_Clear_Info_Fields();
    }
}
else
{
    // Didn't find the tag
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText01 + num_orig_ite
"\p", false );
}

// reset the pointer
data = data_start;
length = data_size;

err = do_Find_Tag( &data, &length, 'orig' );
if( err == noErr )
{
    SInt32 orig = *((SInt32 *)data);
    if( orig == -1 )
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText02 +
num_orig_items, "\pNew Profile", false );
    else
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText02 +
num_orig_items, "\pExisting Profile", false );
    }
else
{
    // Didn't find the tag
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText02 + num_orig_ite
"\p", false );
}

// reset the pointer
data = data_start;
length = data_size;

err = do_Find_Tag( &data, &length, 'dstp' );
if( err == noErr )
{
    SInt32 dstp = *((SInt32 *)data);
    globals->display_type = dstp;
    if( dstp == kDisplayTypeCRT )
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 +
num_orig_items, "\pCRT", false );
    else if( dstp == kDisplayTypeLCD )
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 +
num_orig_items, "\pLCD", false );
    else if( dstp == kDisplayTypeProjector )
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 +
num_orig_items, "\pProjector", false );
    else
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 +
num_orig_items, "\pUnknown", false );
    }
else
{
    // Didn't find the tag
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 + num_orig_ite
"\p", false );
}

// reset the pointer
data = data_start;
length = data_size;

err = do_Find_Tag( &data, &length, 'ctrl' );
if( err == noErr )
{
    SInt32 quantity = *((SInt32 *)data);
    data += sizeof(SInt32);

    if( quantity < 1 )
    {
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 +
num_orig_items, "\pNone", false );
    }
    else if( quantity > 2 )
    {
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 +
num_orig_items, "\pUnknown", false );
    }
    else if( quantity == 1 )
    {
        SInt32 ctrl = *((SInt32 *)data);
```

```
        data += sizeof(SInt32);
        SInt32 val = *((SInt32 *)data);
        data += sizeof(SInt32);

        unsigned char    ctrl_pstr[256];
        ctrl_pstr[0] = 0;

        if( ctrl == kDisplayControlBlackLevel )
        {
            do_p_strcat( ctrl_pstr, "\pBright: " );
            globals->controls_type = kDisplayControlsBrightnessOnly;
        }
        else if( ctrl == kDisplayControlPicture )
        {
            do_p_strcat( ctrl_pstr, "\pContrast: " );
            globals->controls_type = kDisplayControlsContrastOnly;
        }
        else
        {
            do_p_strcat( ctrl_pstr, "\pUnknown: " );
        }

        do_p_strerrcat( ctrl_pstr, val );

        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 +
num_orig_items, ctrl_pstr, false );
    }
    else if( quantity == 2 )
    {
        SInt32 ctrl1 = *((SInt32 *)data);
        data += sizeof(SInt32);
        SInt32 val1 = *((SInt32 *)data);
        data += sizeof(SInt32);
        SInt32 ctrl2 = *((SInt32 *)data);
        data += sizeof(SInt32);
        SInt32 val2 = *((SInt32 *)data);
        data += sizeof(SInt32);

        unsigned char    ctrl_pstr[256];
        ctrl_pstr[0] = 0;

        globals->controls_type = kDisplayControlsBrightnessAndContrast;

        if( ctrl1 == kDisplayControlBlackLevel )
            do_p_strcat( ctrl_pstr, "\pBright: " );
        else if( ctrl1 == kDisplayControlPicture )
            do_p_strcat( ctrl_pstr, "\pContrast: " );
        else
            do_p_strcat( ctrl_pstr, "\pUnknown: " );

        do_p_strerrcat( ctrl_pstr, val1 );

        if( ctrl2 == kDisplayControlBlackLevel )
            do_p_strcat( ctrl_pstr, "\p Bright: " );
        else if( ctrl2 == kDisplayControlPicture )
            do_p_strcat( ctrl_pstr, "\p Contrast: " );
        else
            do_p_strcat( ctrl_pstr, "\p Unknown: " );

        do_p_strerrcat( ctrl_pstr, val2 );

        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 +
num_orig_items, ctrl_pstr, false );
    }
    else
    {
        // Didn't find the tag
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 + num_orig_ite
"\p", false );
    }

    // reset the pointer
    data = data_start;
    length = data_size;

    err = do_Find_Tag( &data, &length, 'reso' );
    if( err == noErr )
    {
        SInt32 horiz = *((SInt32 *)data);
        data += sizeof(SInt32);
        SInt32 vert = *((SInt32 *)data);
        data += sizeof(SInt32);

        unsigned char    reso_pstr[256];
        reso_pstr[0] = 0;

        do_p_strerrcat( reso_pstr, horiz );
```

```
do_p_strcat( reso_pstr, "\p x " );
do_p_strerrcat( reso_pstr, vert );

do_Set_Text_Of_Ditem_As_PString ( window_ref, kNewOrAdjustPaneText05 + num_orig_ite
reso_pstr, false );
}
else
{
    // Didn't find the tag
    do_Set_Text_Of_Ditem_As_PString ( window_ref, kNewOrAdjustPaneText05 + num_orig_ite
"\p", false );
}

// reset the pointer
data = data_start;
length = data_size;

err = do_Find_Tag( &data, &length, 'bitd' );
if( err == noErr )
{
    SInt32 supports_color = *((SInt32 *)data);
    data += sizeof(SInt32);
    SInt32 depth = *((SInt32 *)data);
    data += sizeof(SInt32);

    unsigned char bitd_pstr[256];
    bitd_pstr[0] = 0;

    do_p_strerrcat( bitd_pstr, depth );
    do_p_strcat( bitd_pstr, "\p bit" );

    if( supports_color == 0 )
        do_p_strcat( bitd_pstr, "\p Grayscale " );
    else
        do_p_strcat( bitd_pstr, "\p Color " );

    do_Set_Text_Of_Ditem_As_PString ( window_ref, kNewOrAdjustPaneText06 + num_orig_ite
bitd_pstr, false );
}
else
{
    // Didn't find the tag
    do_Set_Text_Of_Ditem_As_PString ( window_ref, kNewOrAdjustPaneText06 + num_orig_ite
"\p", false );
}

// reset the pointer
data = data_start;
length = data_size;

err = do_Find_Tag( &data, &length, 'freq' );
if( err == noErr )
{
    SInt32 freq = *((SInt32 *)data);
    if( freq != 0 )
        do_Set_Text_Of_Ditem_As_Int ( window_ref, kNewOrAdjustPaneText07 + num_orig_ite
freq, false );
    else
        do_Set_Text_Of_Ditem_As_PString ( window_ref, kNewOrAdjustPaneText07 +
num_orig_items, "\p", false );
}
else
{
    // Didn't find the tag
    do_Set_Text_Of_Ditem_As_PString ( window_ref, kNewOrAdjustPaneText07 + num_orig_ite
"\p", false );
}

// reset the pointer
data = data_start;
length = data_size;

err = do_Find_Tag( &data, &length, 'date' );
if( err == noErr )
{
    struct tm date_tm;
    char time_str[256];
    date_tm = *((struct tm*)data);
    strftime(time_str, (size_t)255, "%b %d, %Y %I:%M:%S %p", &date_tm);
    //
    strftime(time_str, (size_t)255, "%b %d, %Y - %I:%M %p", &date_tm);
    do_Set_Text_Of_Ditem_As_CString ( window_ref, kNewOrAdjustPaneText08 + num_orig_ite
time_str, false );
}
else
{
    // Didn't find the tag
    do_Set_Text_Of_Ditem_As_PString ( window_ref, kNewOrAdjustPaneText08 + num_orig_ite
"\p", false );
}
```

```
    }

    // reset the pointer
    data = data_start;
    length = data_size;

    err = do_Find_Tag( &data, &length, 'gama' );
    if( err == noErr )
    {
        float gamma = *((float *)data);
        if( gamma < 1.0 )
        {
            globals->target_perceptual = true;
            globals->target_gamma = 1.8;
            do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText09 +
num_orig_items, "\pPerceptual", false );
        }
        else
        {
            globals->target_perceptual = false;
            globals->target_gamma = gamma;
            do_Set_Text_Of_DItem_As_Float ( window_ref, kNewOrAdjustPaneText09 + num_orig_i
gamma, false );
        }
    }
    else
    {
        // Didn't find the tag
        do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText09 + num_orig_ite
"\p", false );
    }

    do_Save_Info_Fields();
}
else
{
    do_One_Button_Alert ( kAlertNoteAlert, "\pThe selected profile is not compatible with t
version of SuperCal.", NULL, "\pOK" );
    do_Clear_Info_Fields();
    do_Set_Value_Of_DItem ( window_ref, kNewOrAdjustPaneProfileMenu + num_orig_items,
kProfileMenuNewProfileItem );
    err = -1;
}
}

    free( elem_data );
}
else
{
    err = memFullErr;
}
}

    }

    CMCloseProfile( prof_ref );
}

return( err );
}

//
short o_new_or_adjust_pane::do_Menu_Item_To_Profile_Index( short menu_item )
{
    if( menu_item < kProfileMenuFirstProfileItem ) // kProfileMenuNewProfileItem or kProfileMenuSeparatorItem
    {
        return( -1 );
    }
    else
    {
        return( menu_item - kProfileMenuFirstProfileItem );
    }
}

//
short o_new_or_adjust_pane::do_Profile_Index_To_Menu_Item( long profile_index )
{
    if( profile_index < 0 )
    {
        return( kProfileMenuNewProfileItem );
    }
    else
    {
        return( profile_index + kProfileMenuFirstProfileItem );
    }
}
```

```
//  
// On entry, data is a pointer to the data.  
// On exit, data is a pointer to the beginning byte of the desired tag, if found.  
// On exit, data_length contains the length of the desired data, in bytes.  
  
OSErr o_new_or_adjust_pane::do_Find_Tag ( char **data, UInt32 *data_length, UInt32 desired_tag )  
{  
    OSErr      err = paramErr;  
    UInt32      offset = 0;  
    UInt32      tag = 0;  
    UInt32      tag_length = 0;  
  
    do  
    {  
        tag = *((UInt32 *) (*data + offset));  
        offset += sizeof(UInt32);  
        tag_length = *((UInt32 *) (*data + offset));  
        offset += sizeof(UInt32);  
  
        if( tag == desired_tag )  
            err = noErr;  
        else  
            offset += tag_length;  
    }  
    while ( ( tag != desired_tag ) && ( offset < *data_length ) );  
  
    if( err == noErr )  
    {  
        *data += offset;  
        *data_length = tag_length;  
    }  
  
    return( err );  
}  
  
//  
  
Boolean o_new_or_adjust_pane::do_Validate_Tag ( char *data )  
{  
    Boolean valid = false;  
  
    switch( *((UInt32*)data) )  
    {  
        case 'chan':  
        case 'orig':  
        case 'dstp':  
        case 'ctrl':  
        case 'reso':  
        case 'freq':  
        case 'bitd':  
        case 'date':  
        case 'edid':  
        case 'gama':  
        case 'rimd':  
        {  
            valid = true;  
            break;  
        }  
        default:  
        {  
            valid = false;  
            break;  
        }  
    }  
  
    return( valid );  
}  
  
//  
  
void o_new_or_adjust_pane::do_Clear_Info_Fields ()  
{  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText01 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText02 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText05 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText06 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText07 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText08 + num_orig_items, "\p", false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText09 + num_orig_items, "\p", false );  
  
    do_Save_Info_Fields();  
}  
  
void o_new_or_adjust_pane::do_Save_Info_Fields ()  
{  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText01 + num_orig_items,
```

```
globals->pane_text.NewOrAdjustPaneText01 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText02 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText02 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText03 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText04 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText05 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText05 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText06 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText06 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText07 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText07 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText08 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText08 );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText09 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText09 );  
}  
  
void o_new_or_adjust_pane::do_Restore_Info_Fields ()  
{  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText01 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText01, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText02 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText02, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText03 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText03, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText04 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText04, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText05 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText05, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText06 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText06, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText07 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText07, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText08 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText08, false );  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kNewOrAdjustPaneText09 + num_orig_items,  
globals->pane_text.NewOrAdjustPaneText09, false );  
}
```

```
//  
//  
//
```

©1998-2001 bergdesign inc.

```
#ifndef __o_display_type_pane__  
#define __o_display_type_pane__
```

```
#include "o_base_asst_pane.h"
```

```
class o_display_type_pane : public o_base_asst_pane  
{
```

```
    public:
```

```
        o_display_type_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );
```

```
        void
```

```
        do_Item_Hit ( short );
```

```
    protected:
```

```
    private:
```

```
};
```

```
#endif /* __o_display_type_pane__ */
```

```
//  
// _____ ©1998-2001 bergdesign inc.  
//  
  
#include "o_display_type_pane.h"  
  
enum  
{  
    kDisplayTypePaneDITL          = 3400,  
    kDisplayTypePaneAppendMode    = -1  
};  
  
enum  
{  
    kDisplayTypePaneStaticText1    = 1,  
    kDisplayTypePaneStaticText2    = 2,  
    kDisplayTypePaneCRTPicture     = 3,  
    kDisplayTypePaneCRTRadioButton = 4,  
    kDisplayTypePaneLCDPicture     = 5,  
    kDisplayTypePaneLCDRadioButton = 6,  
    kDisplayTypePaneProjectorPicture = 7,  
    kDisplayTypePaneProjectorRadioButton = 8  
};  
  
//  
  
o_display_type_pane::o_display_type_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kDisplayTypePaneDITL, kDisplayTypePaneAppendMode, cal_globals )  
{  
    if ( globals->display_type == kDisplayTypeCRT )  
    {  
        do_Item_Hit ( kDisplayTypePaneCRTRadioButton + num_orig_items );  
    }  
    else if ( globals->display_type == kDisplayTypeLCD )  
    {  
        do_Item_Hit ( kDisplayTypePaneLCDRadioButton + num_orig_items );  
    }  
    else if ( globals->display_type == kDisplayTypeProjector )  
    {  
        do_Item_Hit ( kDisplayTypePaneProjectorRadioButton + num_orig_items );  
    }  
    else  
    {  
        do_Item_Hit ( kDisplayTypePaneLCDRadioButton + num_orig_items );  
    }  
}  
  
//  
  
void  
o_display_type_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kDisplayTypePaneCRTPicture:  
        case kDisplayTypePaneCRTRadioButton:  
        {  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneCRTRadioButton + num_orig_items, 1 );  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneLCDRadioButton + num_orig_items, 0 );  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneProjectorRadioButton + num_orig_items, 0 );  
        };  
        globals->display_type = kDisplayTypeCRT;  
        break;  
    }  
    case kDisplayTypePaneLCDPicture:  
    case kDisplayTypePaneLCDRadioButton:  
    {  
        do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneCRTRadioButton + num_orig_items, 0 );  
        do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneLCDRadioButton + num_orig_items, 1 );  
        do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneProjectorRadioButton + num_orig_items, 0 );  
    };  
    globals->display_type = kDisplayTypeLCD;  
    break;  
    }  
    case kDisplayTypePaneProjectorPicture:  
    case kDisplayTypePaneProjectorRadioButton:  
    {  
        do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneCRTRadioButton + num_orig_items, 0 );  
        do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneLCDRadioButton + num_orig_items, 0 );  
        do_Set_Value_Of_DItem_As_Boolean ( window_ref, kDisplayTypePaneProjectorRadioButton + num_orig_items, 1 );  
    };  
    globals->display_type = kDisplayTypeProjector;  
    break;  
    }  
    default:  
    {  
    }  
}
```



```
    {  
        break;  
    }  
}
```

```
//  
//  
//
```

```
#ifndef __o_control_type_pane__  
#define __o_control_type_pane__
```

```
#include "o_base_asst_pane.h"
```

```
class o_control_type_pane : public o_base_asst_pane  
{
```

```
    public:
```

```
        o_control_type_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );
```

```
        void do_Item_Hit ( short );
```

```
    protected:
```

```
    private:
```

```
};
```

```
#endif /* __o_control_type_pane__ */
```

```
// _____  
//                               ©1998-2001 bergdesign inc.  
// _____  
  
#include "o_control_type_pane.h"  
  
enum  
{  
    kControlTypePaneDITL          = 3200,  
    kControlTypePaneAppendMode    = -1  
};  
  
enum  
{  
    kControlTypePaneStaticText1    = 1,  
    kControlTypePaneBAndCRadioButton = 2,  
    kControlTypePaneBAndCPicture    = 3,  
    kControlTypePaneBOnlyRadioButton = 4,  
    kControlTypePaneBOnlyPicture     = 5,  
    kControlTypePaneCOnlyRadioButton = 6,  
    kControlTypePaneCOnlyPicture     = 7  
};  
  
// _____  
  
o_control_type_pane::o_control_type_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kControlTypePaneDITL, kControlTypePaneAppendMode, cal_globals )  
{  
    if ( globals->controls_type )  
    {  
        if ( globals->controls_type == kDisplayControlsBrightnessAndContrast )  
        {  
            do_Item_Hit ( kControlTypePaneBAndCRadioButton + num_orig_items );  
        }  
        else if ( globals->controls_type == kDisplayControlsBrightnessOnly )  
        {  
            do_Item_Hit ( kControlTypePaneBOnlyRadioButton + num_orig_items );  
        }  
        else // kDisplayControlsContrastOnly  
        {  
            do_Item_Hit ( kControlTypePaneCOnlyRadioButton + num_orig_items );  
        }  
    }  
    else  
    {  
        if ( globals->display_type == kDisplayTypeLCD )  
        {  
            do_Item_Hit ( kControlTypePaneBOnlyRadioButton + num_orig_items );  
        }  
        else // kDisplayTypeCRT or kDisplayTypeProjector  
        {  
            do_Item_Hit ( kControlTypePaneBAndCRadioButton + num_orig_items );  
        }  
    }  
}  
  
// _____  
  
void  
o_control_type_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kControlTypePaneBAndCRadioButton:  
        case kControlTypePaneBAndCPicture:  
        {  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneBAndCRadioButton + num_orig_items, 1 );  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneBOnlyRadioButton + num_orig_items, 0 );  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneCOnlyRadioButton + num_orig_items, 0 );  
            globals->controls_type = kDisplayControlsBrightnessAndContrast;  
            break;  
        }  
        case kControlTypePaneBOnlyRadioButton:  
        case kControlTypePaneBOnlyPicture:  
        {  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneBAndCRadioButton + num_orig_items, 0 );  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneBOnlyRadioButton + num_orig_items, 1 );  
            do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneCOnlyRadioButton + num_orig_items, 0 );  
            globals->controls_type = kDisplayControlsBrightnessOnly;  
            break;  
        }  
        case kControlTypePaneCOnlyRadioButton:  
        case kControlTypePaneCOnlyPicture:  
        {  

```

```
do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneBAndCRadioButton + num_orig_items, 0 );  
do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneBOnlyRadioButton + num_orig_items, 0 );  
do_Set_Value_Of_DItem_As_Boolean ( window_ref, kControlTypePaneCOnlyRadioButton + num_orig_items, 1 );  
globals->controls_type = kDisplayControlsContrastOnly;  
break;  
}  
default:  
{  
    break;  
}  
}  
}
```

```
//  
//  
//
```

```
#ifndef __o_adjust_display_pane__  
#define __o_adjust_display_pane__
```

```
#include "o_base_asst_pane.h"  
#include "o_help_pane.h"  
#include "o_black_window.h"
```

```
class o_adjust_display_pane : public o_base_asst_pane  
{
```

```
    public:
```

```
        o_adjust_display_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );  
        ~o_adjust_display_pane ();
```

```
        void do_Item_Hit ( short );
```

```
    protected:
```

```
    private:
```

```
        o_help_pane *help_pane;
```

```
};
```

```
#endif /* __o_adjust_display_pane__ */
```



```
k_Projector_B_and_C_Pict );
    }
}
else if ( globals->controls_type == kDisplayControlsBrightnessOnly )
{
    HideDialogItem ( window_ref, kAdjustDisplayPaneBlackButton + num_orig_items );

    if ( globals->display_type == kDisplayTypeCRT )
    {
        help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),
                                          kAdjustDisplayPanePlaceholder + num_orig_items,
                                          k_CRT_B_only_NumSteps, k_CRT_B_only_Text, k_CRT_B_only_Pict );
    }
    else if ( globals->display_type == kDisplayTypeLCD )
    {
        help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),
                                          kAdjustDisplayPanePlaceholder + num_orig_items,
                                          k_LCD_B_only_NumSteps, k_LCD_B_only_Text, k_LCD_B_only_Pict );
    }
    else // if ( globals->display_type == kDisplayTypeProjector )
    {
        help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),
                                          kAdjustDisplayPanePlaceholder + num_orig_items,
                                          k_Projector_B_only_NumSteps, k_Projector_B_only_Text, k_Projector_B_onl
);
    }
}
else // globals->controls_type == kDisplayControlsContrastOnly
{
    HideDialogItem ( window_ref, kAdjustDisplayPaneBlackButton + num_orig_items );

    if ( globals->display_type == kDisplayTypeCRT )
    {
        help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),
                                          kAdjustDisplayPanePlaceholder + num_orig_items,
                                          k_CRT_C_only_NumSteps, k_CRT_C_only_Text, k_CRT_C_only_Pict );
    }
    else if ( globals->display_type == kDisplayTypeLCD )
    {
        help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),
                                          kAdjustDisplayPanePlaceholder + num_orig_items,
                                          k_LCD_C_only_NumSteps, k_LCD_C_only_Text, k_LCD_C_only_Pict );
    }
    else // if ( globals->display_type == kDisplayTypeProjector )
    {
        help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),
                                          kAdjustDisplayPanePlaceholder + num_orig_items,
                                          k_Projector_C_only_NumSteps, k_Projector_C_only_Text, k_Projector_C_onl
);
    }
}
}
//
o_adjust_display_pane::~o_adjust_display_pane()
{
    if ( help_pane != NULL )
        delete help_pane;
}
//
void o_adjust_display_pane::do_Item_Hit ( short item_hit )
{
    switch ( item_hit - num_orig_items )
    {
        case kAdjustDisplayPaneBlackButton:
        {
            if ( globals->black_window == NULL )
            {
                if ( GetMBarHeight() != 0 )
                    do_Show_Menu_Bar ( false );
                else
                    do_Show_Menu_Bar ( true );

                //
                globals->black_window = new o_black_window ( &globals->display_bounds, true,
(WindowAttributes)kHasNoTitleBar, kThemeBrushChasingArrows, (WindowRef)-1, globals );
                globals->black_window = new o_black_window ( &globals->display_bounds, true,
(WindowAttributes)kWindowNoAttributes, kThemeBrushChasingArrows, (WindowRef)-1, globals );
            }

            break;
        }
        case kAdjustDisplayPaneWhiteButton:
        {
            if ( globals->black_window == NULL )
            {

```

```
        if ( GetMBarHeight() != 0 )
            do_Show_Menu_Bar ( false );
        else
            do_Show_Menu_Bar ( true );

//        globals->black_window = new o_black_window ( &globals->display_bounds, true,
(WindowAttributes)kHasNoTitlebar, kThemeBrushDocumentWindowBackground, (WindowRef)-1, globals );
        globals->black_window = new o_black_window ( &globals->display_bounds, true,
(WindowAttributes)kWindowNoAttributes, kThemeBrushDocumentWindowBackground, (WindowRef)-1, globals );
    }

    break;
}
default:
{
    help_pane->do_Item_Hit ( item_hit );
    break;
}
}
}
```

```
//  
//  
//
```

```
#ifndef __o_black_level_pane__  
#define __o_black_level_pane__
```

```
#include "o_base_asst_pane.h"  
#include "o_help_pane.h"  
#include "o_black_level_window.h"
```

```
class o_black_level_pane : public o_base_asst_pane  
{  
    public:  
        o_black_level_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );  
        ~o_black_level_pane ();  
  
        void do_Item_Hit ( short );  
        void do_Update ();  
  
    protected:  
  
    private:  
        o_help_pane *help_pane;  
};  
  
#endif /* __o_black_level_pane__ */
```

```
//  
// _____ ©1998-2001 bergdesign inc.  
//  
  
#include "o_black_level_pane.h"  
  
enum  
{  
    kBlackLevelPaneDITL          = 3700,  
    kBlackLevelPaneAppendMode    = -1  
};  
  
enum  
{  
    kBlackLevelStaticText1       = 1,  
    kBlackLevelMeasureButton     = 2,  
    kBlackLevelHelpPlaceholder   = 3  
};  
  
enum  
{  
    kBlackLevelHelpNumSteps      = 3,  
    kBlackLevelHelpText          = 3200,  
    kBlackLevelHelpPict          = 3200  
};  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
o_black_level_pane::o_black_level_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kBlackLevelPaneDITL, kBlackLevelPaneAppendMode, cal_globals )  
{  
    help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),  
                                    kBlackLevelHelpPlaceholder + num_orig_items,  
                                    kBlackLevelHelpNumSteps, kBlackLevelHelpText, kBlackLevelHelpPict );  
}  
  
// _____  
o_black_level_pane::~o_black_level_pane ()  
{  
    if ( help_pane != NULL )  
        delete help_pane;  
}  
  
// _____  
void  
o_black_level_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kBlackLevelMeasureButton:  
        {  
            if( globals->black_level_window == NULL )  
            {  
                if ( globals->response_complete )  
                {  
                    item_hit = do_Two_Button_Alert (    kAlertCautionAlert,  
                                                         "\pMeasuring the black level will invalidate the response  
measurement. Do you want to continue?",  
                                                         "\pThis action cannot be undone.",  
                                                         "\pContinue",  
                                                         "\pCancel" );  
  
                    if ( item_hit == kAlertStdAlertOKButton )  
                    {  
                        // If the user chose to continue, we need to invalidate the response measurement.  
                        globals->response_complete = false;  
                        globals->white_point_complete = false;  
                    }  
                    else // item_hit == kAlertStdAlertCancelButton  
                    {  
                        break;  
                    }  
                }  
            }  
  
            globals->black_level_window = new o_black_level_window ( &globals->display_bounds, true,  
(WindowAttributes)kWindowNoAttributes, kThemeChasingArrowsBrush, (WindowRef)-1, globals );  
            break;  
        }  
        default:  
        {  
            help_pane->do_Item_Hit ( item_hit );  
        }  
    }  
}
```

```
        break;
    }
}

//
void o_black_level_pane::do_Update ()
{
    DEBUG_PRINT("o_black_level_pane::do_Update()");

    Str255 title;
    do_Get_Title_Of_DItem( window_ref, kBlackLevelMeasureButton + num_orig_items, title );

    if( globals->black_level_window == NULL )
        do_Activate_DItem ( window_ref, kBlackLevelMeasureButton + num_orig_items, true );
    else
        do_Activate_DItem ( window_ref, kBlackLevelMeasureButton + num_orig_items, false );

    if ( globals->black_level_complete )
    {
        if( do_ci_p_strcmp( title, "\pRe-Measure" ) )
            do_Set_Title_Of_DItem ( window_ref, kBlackLevelMeasureButton + num_orig_items, "\pRe-Measure" );
    }
    else
    {
        if( do_ci_p_strcmp( title, "\pBegin Measuring" ) )
            do_Set_Title_Of_DItem ( window_ref, kBlackLevelMeasureButton + num_orig_items, "\pBegin Measuring" );
    }
}
```



```
// _____  
//                               ©1998-2002 bergdesign inc.  
// _____  
  
#include "o_response_pane.h"  
  
enum  
{  
    kGammaPaneDITL          = 4000,  
    kGammaPaneAppendMode    = -1  
};  
  
enum  
{  
    kGammaPaneStaticText1    = 1,  
    kGammaPaneMeasureButton  = 2,  
    kGammaPaneHelpPlaceholder = 3  
};  
  
enum  
{  
    kGammaPaneHelpNumSteps    = 7,  
    kGammaPaneHelpText        = 3300,  
    kGammaPaneHelpPict        = 3300  
};  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
o_response_pane::o_response_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kGammaPaneDITL, kGammaPaneAppendMode, cal_globals )  
{  
    help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),  
                                    kGammaPaneHelpPlaceholder + num_orig_items,  
                                    kGammaPaneHelpNumSteps, kGammaPaneHelpText, kGammaPaneHelpPict );  
  
    copy_gamma_to_dev( globals->this_component.saved_dev_info );  
}  
  
// _____  
o_response_pane::~o_response_pane ()  
{  
    if ( help_pane != NULL )  
        delete help_pane;  
}  
  
// _____  
void  
o_response_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kGammaPaneMeasureButton:  
        {  
            if( globals->response_window == NULL )  
                globals->response_window = new o_response_window ( &globals->display_bounds, true,  
(WindowAttributes)kWindowNoAttributes, kThemeChasingArrowsBrush, (WindowRef)-1, globals );  
            break;  
        }  
        default:  
        {  
            help_pane->do_Item_Hit ( item_hit );  
            break;  
        }  
    }  
}  
  
// _____  
void o_response_pane::do_Update ()  
{  
    DEBUG_PRINT("o_response_pane::do_Update()");  
  
    Str255 title;  
    do_Get_Title_Of_DItem( window_ref, kGammaPaneMeasureButton + num_orig_items, title );  
  
    if ( globals->black_level_complete && globals->response_window == NULL )  
        do_Activate_DItem ( window_ref, kGammaPaneMeasureButton + num_orig_items, true );  
    else  
        do_Activate_DItem ( window_ref, kGammaPaneMeasureButton + num_orig_items, false );  
  
    if ( globals->response_complete )
```

```
{
    if( do_ci_p_strcmp( title, "\pRe-Measure" ) )
        do_Set_Title_Of_DItem ( window_ref, kGammaPaneMeasureButton + num_orig_items, "\pRe-Measure" );
    }
    else
    {
        if( do_ci_p_strcmp( title, "\pBegin Measuring" ) )
            do_Set_Title_Of_DItem ( window_ref, kGammaPaneMeasureButton + num_orig_items, "\pBegin Measuring" );
        }
    }
}
```

```
//  
//  
//
```

```
#ifndef __o_white_point_pane__  
#define __o_white_point_pane__
```

```
#include "o_base_asst_pane.h"  
#include "o_help_pane.h"  
#include "o_white_point_window.h"  
#include "o_viewer_window.h"
```

```
class o_white_point_pane : public o_base_asst_pane  
{
```

```
    public:  
        o_white_point_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );  
        ~o_white_point_pane ();
```

```
        void do_Item_Hit ( short );  
        void do_Update ();
```

```
    protected:
```

```
    private:  
        o_help_pane *help_pane;
```

```
};
```

```
#endif /* __o_white_point_pane__ */
```

```
//  
//  
//  
#include "o_white_point_pane.h"  
  
enum  
{  
    kWhitePointPaneDITL          = 4300,  
    kWhitePointPaneAppendMode    = -1  
};  
  
enum  
{  
    kWhitePointStaticText1       = 1,  
    kWhitePointMeasureButton     = 2,  
    kWhitePointViewButton        = 3,  
    kWhitePointHelpPlaceholder   = 4  
};  
  
enum  
{  
    kWhitePointHelpNumSteps      = 3,  
    kWhitePointHelpText          = 3400,  
    kWhitePointHelpPict          = 3700  
};  
  
//  
o_white_point_pane::o_white_point_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kWhitePointPaneDITL, kWhitePointPaneAppendMode, cal_globals )  
{  
    help_pane = new o_help_pane (    window_ref, CountDITL(window_ref),  
                                     kWhitePointHelpPlaceholder + num_orig_items,  
                                     kWhitePointHelpNumSteps, kWhitePointHelpText, kWhitePointHelpPict );  
  
    if( globals->response_complete )  
    {  
        control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header,  
globals->target_perceptual, globals->target_gamma, &(globals->this_component) );  
        copy_gamma_to_dev( globals->this_component.this_dev_info );  
    }  
}  
  
//  
o_white_point_pane::~o_white_point_pane ()  
{  
    if ( help_pane != NULL )  
        delete help_pane;  
}  
  
//  
  
void  
o_white_point_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kWhitePointMeasureButton:  
        {  
            if( globals->white_point_window == NULL )  
                globals->white_point_window = new o_white_point_window ( &globals->display_bounds, true,  
(WindowAttributes)kWindowNoAttributes, kThemeChasingArrowsBrush, (WindowRef)-1, globals );  
//            globals->white_point_window = new o_white_point_window ( &globals->display_bounds, true,  
(WindowAttributes)kWindowNoAttributes, kThemeDocumentWindowBackgroundBrush, (WindowRef)-1, globals );  
  
            break;  
        }  
        case kWhitePointViewButton:  
        {  
            if( globals->viewer_window == NULL )  
                globals->viewer_window = new o_viewer_window ( &globals->display_bounds, true,  
(WindowAttributes)kWindowNoAttributes, kThemeChasingArrowsBrush, (WindowRef)-1, globals );  
  
            break;  
        }  
        default:  
        {  
            help_pane->do_Item_Hit ( item_hit );  
            break;  
        }  
    }  
}
```



```
//  
  
void o_white_point_pane::do_Update ()  
{  
    Str255 title;  
    do_Get_Title_Of_DItem( window_ref, kWhitePointMeasureButton + num_orig_items, title );  
  
    if ( globals->response_complete && globals->white_point_window == NULL )  
        do_Activate_DItem ( window_ref, kWhitePointMeasureButton + num_orig_items, true );  
    else  
        do_Activate_DItem ( window_ref, kWhitePointMeasureButton + num_orig_items, false );  
  
    if ( globals->white_point_complete )  
    {  
        if( do_ci_p_strcmp( title, "\pRe-Adjust" ) )  
            do_Set_Title_Of_DItem ( window_ref, kWhitePointMeasureButton + num_orig_items, "\pRe-Adjust" );  
    }  
    else  
    {  
        if( do_ci_p_strcmp( title, "\pAdjust" ) )  
            do_Set_Title_Of_DItem ( window_ref, kWhitePointMeasureButton + num_orig_items, "\pAdjust" );  
    }  
  
    if( globals->response_complete && globals->viewer_window == NULL )  
        do_Activate_DItem ( window_ref, kWhitePointViewButton + num_orig_items, true );  
    else  
        do_Activate_DItem ( window_ref, kWhitePointViewButton + num_orig_items, false );  
}
```



```
{
    globals->target_perceptual = true;
    do_Activate_DItem ( window_ref, kGammaTargetPaneGroupEmbedder + num_orig_items, false );
}

update_screen = true;
break;
}
case kGammaTargetPaneLinearIcon:
{
    do_Set_Value_Of_DItem_As_Clipped ( window_ref, kGammaTargetPaneGammaSlider + num_orig_items, 10 );
    // A bug work-around
    if( do_Check_For_Aqua_Menus() )
        do_Draw_One_Control_As_DItem ( window_ref, kGammaTargetPaneGammaSlider + num_orig_items );
    globals->target_gamma = 1.0;
    update_screen = true;
    break;
}
case kGammaTargetPaneMacIcon:
{
    do_Set_Value_Of_DItem_As_Clipped ( window_ref, kGammaTargetPaneGammaSlider + num_orig_items, 18 );
    globals->target_gamma = 1.8;
    update_screen = true;
    break;
}
case kGammaTargetPaneTVIcon:
{
    do_Set_Value_Of_DItem_As_Clipped ( window_ref, kGammaTargetPaneGammaSlider + num_orig_items, 22 );
    globals->target_gamma = 2.2;
    update_screen = true;
    break;
}
case kGammaTargetPanePCIcon:
{
    do_Set_Value_Of_DItem_As_Clipped ( window_ref, kGammaTargetPaneGammaSlider + num_orig_items, 25 );
    globals->target_gamma = 2.5;
    update_screen = true;
    break;
}
case kGammaTargetPaneGammaSlider:
{
    short value;

    do_Get_Value_Of_DItem ( window_ref, kGammaTargetPaneGammaSlider + num_orig_items, &value );
    globals->target_gamma = (float)value / 10.0;
    update_screen = true;
    break;
}
default:
{
    break;
}
}

if( update_screen == true )
{
    control_points_to_table( globals->this_component.this_dev_info->gamma_table_w_header,
globals->target_perceptual, globals->target_gamma, &(globals->this_component) );
    copy_gamma_to_dev( globals->this_component.this_dev_info );
}
}

//
void o_gamma_target_pane::do_Update ()
{
    if ( globals->response_complete )
    {
        do_Activate_DItem ( window_ref, kGammaTargetPanePopupGroup + num_orig_items, true );
    }
    else
    {
        do_Activate_DItem ( window_ref, kGammaTargetPanePopupGroup + num_orig_items, false );
    }
}
```

```
//  
//  
//
```

```
#ifndef __o_phosphors_pane__  
#define __o_phosphors_pane__
```

```
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Lists.h>  
#endif  
#endif
```

```
#include "o_base_asst_pane.h"
```

```
class o_phosphors_pane : public o_base_asst_pane  
{  
public:  
    o_phosphors_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );  
    ~o_phosphors_pane ();  
  
    void do_Item_Hit ( short );  
  
protected:  
    ListHandle list_handle;  
  
private:  
};  
  
#endif /* __o_phosphors_pane__ */
```

```
//  
//  
//  
#include "o_phosphors_pane.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
enum  
{  
    kPhosphorsPaneDITL          = 3900,  
    kPhosphorsPaneAppendMode    = -1  
  
    // kPhosphorsPaneStringListID = 3900  
};  
  
enum  
{  
    kPhosphorsPaneStaticText    = 1,  
    kPhosphorsPaneListBox      = 2  
};  
  
//  
  
o_phosphors_pane::o_phosphors_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kPhosphorsPaneDITL, kPhosphorsPaneAppendMode, cal_globals )  
{  
    ControlHandle    control;  
    OSErr            err = noErr;  
  
    err = do_Set_Text_Style_Of_DItem ( window_ref, num_orig_items + kPhosphorsPaneListBox, 1, NULL, NULL, NULL );  
    if ( err != noErr )  
    {  
        do_One_Button_Alert ( kAlertStopAlert, "\pThere was an error setting the font of the list box.", NULL,  
            "\pOK" );  
        DEBUG_VAR_PRINT("List box font error = %d",err);  
    }  
  
    GetDialogItemAsControl ( window_ref, num_orig_items + kPhosphorsPaneListBox, &control );  
    err = do_Get_List_Box_List_Handle ( control, &list_handle );  
    if ( err == noErr )  
    {  
        Cell        cell;  
        Str255       string;  
        short        i;  
  
        cell.h = 0;  
  
        // for ( i = 1; true; i++ )  
        // {  
        //     GetIndString ( string, kPhosphorsPaneStringListID, i );  
        //     if ( string[0] == 0 )  
        //         break;  
        // }  
  
        for ( i = 0; i < globals->tri_count; i++ )  
        {  
            LAddRow( 1, (**list_handle).dataBounds.bottom, list_handle );  
            cell.v = (**list_handle).dataBounds.bottom - 1;  
  
            // The p_strcat function will keep the string within length bounds  
            string[0] = 0;  
            do_p_strcat( string, globals->tri_data[i].manufacturer );  
            do_p_strcat( string, "\p " );  
            do_p_strcat( string, globals->tri_data[i].model );  
            do_p_strcat( string, "\p " );  
            do_p_strcat( string, globals->tri_data[i].variant );  
  
            LSetCell( (Ptr)(string + 1), string[0], cell, list_handle );  
        }  
  
        if( globals->tri_choice < globals->tri_count )  
            SetPt( &cell, 0, globals->tri_choice );  
        else  
            SetPt( &cell, 0, 0 );  
  
        LSetSelect( true, cell, list_handle );  
        do_Item_Hit( kPhosphorsPaneListBox + num_orig_items );  
    }  
}  
  
//  
  
o_phosphors_pane::~o_phosphors_pane ()  
{  
    list_handle = NULL;  
}
```

```
}  
  
//  
  
void o_phosphors_pane::do_Item_Hit ( short item_hit )  
{  
    switch ( item_hit - num_orig_items )  
    {  
        case kPhosphorsPaneListBox:  
        {  
            Cell    the_cell;  
            SetPt( &the_cell, 0, 0 );  
            Boolean found = LGetSelect( true, &the_cell, list_handle );  
            if( found == true )  
            {  
                globals->tri_choice = the_cell.v;  
                do_Alert_If_Error( "\pListBox value = ", the_cell.v );  
                do_Get_Value_Of_DItem ( window_ref, kPhosphorsPaneListBox + num_orig_items, &val );  
            }  
            break;  
        }  
        default:  
        {  
            break;  
        }  
    }  
}
```

```
//  
//  
//  
#ifndef __o_save_profile_pane__  
#define __o_save_profile_pane__  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#include <QuickTime/ImageCompression.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#include <ImageCompression.h>  
#else  
#include <Folders.h>  
#include <Displays.h>  
#endif  
#endif  
  
#include "o_base_asst_pane.h"  
#include "cal_math.h"  
#include "my_utilities.h"  
#include "my_gestalts.h"  
#include "my_colorsync.h"  
#include "my_displays.h"  
  
class o_save_profile_pane : public o_base_asst_pane  
{  
public:  
    o_save_profile_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals );  
    ~o_save_profile_pane();  
  
protected:  
    void do_Item_Hit ( short );  
    void do_Key_Down_Post_Processing ();  
    OSError do_Save_Profile ();  
    CMError do_Set_Profile_Description_Tag ( CMPProfileRef, Str255, ScriptCode );  
    CMError do_Set_Profile_Copyright_Tag ( CMPProfileRef, const Str255, ScriptCode );  
    CMError do_Get_Profile_Colorant_Tag ( CMPProfileRef, OSType, CMFixedXYZColor * );  
    CMError do_Set_Profile_Colorant_Tag ( CMPProfileRef, OSType, CMFixedXYZColor * );  
    CMError do_Set_Profile_TRC_Tag( CMPProfileRef, OSType, float );  
    CMError do_Set_Profile_VCGamma_Tag ( CMPProfileRef );  
    CMError do_Set_Profile_Custom_Measurements_Tag ( CMPProfileRef );  
  
private:  
    static pascal ControlKeyFilterResult do_Edit_Text_Field_Filter ( ControlHandle, short *, short *, short  
);  
    ControlKeyFilterUPP edit_text_filter_proc;  
};  
  
#endif /* __o_save_profile_pane__ */
```



```
//  
// _____ ©1998-2001 bergdesign inc.  
//  
  
#include "o_save_profile_pane.h"  
  
enum  
{  
    kSaveProfilePaneDITL          = 4200,  
    kSaveProfilePaneAppendMode    = -1  
};  
  
enum  
{  
    kSaveProfilePaneStaticText1    = 1,  
    kSaveProfilePaneStaticText2    = 2,  
    kSaveProfilePaneEditTextFileName = 3,  
    kSaveProfilePaneStaticTextFileName = 4,  
    kSaveProfilePaneStaticTextFileNameInfo = 5,  
    kSaveProfilePaneEditTextCSName = 6,  
    kSaveProfilePaneStaticTextCSName = 7,  
    kSaveProfilePaneStaticTextCSNameInfo = 8,  
    kSaveProfilePaneSaveProfileButton = 9  
};  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
o_save_profile_pane::o_save_profile_pane ( DialogRef dialog, short items, struct cal_globals *cal_globals )  
    : o_base_asst_pane ( dialog, items, kSaveProfilePaneDITL, kSaveProfilePaneAppendMode, cal_globals )  
{  
    edit_text_filter_proc = NewControlKeyFilterUPP (   
    (ControlKeyFilterProcPtr)o_save_profile_pane::do_Edit_Text_Field_Filter );  
  
    do_Set_Key_Filter_Of_DItem ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items,  
    edit_text_filter_proc );  
    do_Set_Key_Filter_Of_DItem ( window_ref, kSaveProfilePaneEditTextCSName + num_orig_items, edit_text_filter_proc );  
  
    do_Set_Control_Ref_Of_DItem ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items, (long)this );  
    do_Set_Control_Ref_Of_DItem ( window_ref, kSaveProfilePaneEditTextCSName + num_orig_items, (long)this );  
  
    OSStatus err = do_Set_Keyboard_Focus_As_DItem ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items );  
};  
// ClearKeyboardFocus ( (WindowPtr>window_ref );  
  
if( globals->profile_file_name[0] != 0 )  
{  
    do_Set_Text_Of_DItem_As_PString ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items,  
    globals->profile_file_name, true );  
  
    if( true == globals->create_profile )  
        do_Activate_DItem ( window_ref, kSaveProfilePaneSaveProfileButton + num_orig_items, false );  
    else  
    {  
        do_Activate_DItem ( window_ref, kSaveProfilePaneSaveProfileButton + num_orig_items, false );  
    }  
  
    if( globals->profile_name[0] != 0 )  
    {  
        do_Set_Text_Of_DItem_As_PString ( window_ref, kSaveProfilePaneEditTextCSName + num_orig_items,  
        globals->profile_name, false );  
    }  
}  
  
// _____  
  
o_save_profile_pane::~o_save_profile_pane ()  
{  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items,  
    globals->profile_file_name );  
    do_Get_Text_Of_DItem_As_PString ( window_ref, kSaveProfilePaneEditTextCSName + num_orig_items,  
    globals->profile_name );  
  
    if( edit_text_filter_proc )  
        DisposeRoutineDescriptor( edit_text_filter_proc );  
    DisposeControlKeyFilterUPP( edit_text_filter_proc );  
}  
  
// _____  
  
void o_save_profile_pane::do_Item_Hit ( short item_hit )  
{  
    short item_with_focus;
```

```
int      err = noErr;

    item_with_focus = do_Get_Keyboard_Focus_As_DItem ( window_ref );

    switch ( item_hit - num_orig_items )
    {
        case kSaveProfilePaneEditTextFileName:
        {
            if( ( item_with_focus - num_orig_items ) != kSaveProfilePaneEditTextFileName )
                do_Set_Keyboard_Focus_As_DItem ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items );

            break;
        }
        case kSaveProfilePaneEditTextCSName:
        {
            if( ( item_with_focus - num_orig_items ) != kSaveProfilePaneEditTextCSName )
                do_Set_Keyboard_Focus_As_DItem ( window_ref, kSaveProfilePaneEditTextCSName + num_orig_items );

            break;
        }
        case kSaveProfilePaneSaveProfileButton:
        {
            err = do_Save_Profile();
            if( !err )
            {
                globals->create_profile = true;
                do_Activate_DItem ( window_ref, kSaveProfilePaneSaveProfileButton + num_orig_items, false );
            }
            break;
        }
        default:
        {
            break;
        }
    }
}

//

void o_save_profile_pane::do_Key_Down_Post_Processing ()
{
    Str255      the_string;

    do_Get_Text_Of_DItem_As_PString ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items, the_string );

    if( the_string[0] == 0 )
    {
        if( do_Is_DItem_Active ( window_ref, kSaveProfilePaneSaveProfileButton + num_orig_items ) )
            do_Activate_DItem ( window_ref, kSaveProfilePaneSaveProfileButton + num_orig_items, false );
    }
    else
    {
        // Also need to check whether black level and response measurements have been done
        if( globals->black_level_complete == true && globals->response_complete == true )
        {
            if( !do_Is_DItem_Active ( window_ref, kSaveProfilePaneSaveProfileButton + num_orig_items ) )
                do_Activate_DItem ( window_ref, kSaveProfilePaneSaveProfileButton + num_orig_items, true );
        }
    }
}

//

OSErr o_save_profile_pane::do_Save_Profile ()
{
    int          err = noErr;
    //CMAppleProfileHeader  header;
    CM2Header    header;
    unsigned long version_offset;
    unsigned char major_rev;
    short        vol_ref_num;
    long         dir_id_num;
    FSSpec       file_spec;
    Str255       file_name;
    Str255       colorsync_name;
    Str255       error_text;
    short        item_hit;
    Boolean       found_it = false;
    CMPProfileRef prof_ref = NULL;
    CMPProfileLocation prof_loc;
    long         prof_count = 0;
    DateTimeRec  date_time;
    CMFixedXYZColor red_color, green_color, blue_color, white_color, black_color;
    int          profile_folder_loc = kOnAppropriateDisk;
    Boolean       work = true;

    // Tag Name          General Description
    // -----

```

```
// profileDescriptionTag    Structure containing invariant and localizable versions of the profile name for
display
// redColorantTag           Relative XYZ values of red phosphor
// greenColorantTag         Relative XYZ values of green phosphor
// blueColorantTag          Relative XYZ values of blue phosphor
// redTRCTag                Red channel tone reproduction curve
// greenTRCTag              Green channel tone reproduction curve
// blueTRCTag               Blue channel tone reproduction curve
// mediaWhitePointTag       Media XYZ white point
// copyrightTag             7 bit ASCII profile copyright information

// Remember - All chromaticities need to be adapted to a D50 temp if the native white point
// is different. Can use the Bradford methods to do the adaptation.

// ColorSync has APIs to do chromatic adaptation.

error_text[0] = 0;

// If we're running under 10 (or CarbonLib 1.3.1 or greater), we use the new
// folder specifier. Otherwise, we'll get a -35 volume not found error.
if( do_Check_For_Aqua_Menus() )
    profile_folder_loc = kUserDomain;
else
    profile_folder_loc = kOnSystemDisk;

err = FindFolder ( profile_folder_loc, 'prof', kCreateFolder, &vol_ref_num, &dir_id_num );
if( err )
{
    do_One_Button_Alert ( kAlertStopAlert, "\pAn error occurred locating the ColorSync profiles folder.", NULL,
"\pOK" );
    goto bail;
}

err = do_Get_Text_Of_DItem_As_PString ( window_ref, kSaveProfilePaneEditTextFileName + num_orig_items,
file_name );
if( err )
    goto bail;

err = do_Get_Text_Of_DItem_As_PString ( window_ref, kSaveProfilePaneEditTextCSName + num_orig_items,
colorsync_name );
if( err )
    goto bail;

// The FSMakeFSSpec() function will determine for us whether the file already exists or not.
err = FSMakeFSSpec ( vol_ref_num, dir_id_num, file_name, &file_spec );
if( err == noErr || err == fnfErr )
{
    prof_loc.locType = cmFileBasedProfile;
    prof_loc.u.fileLoc.spec = file_spec;

    // If there is no error, then a file pointed to by the FSSpec exists.
    // We need to ask the user to replace the file that already exists.
    if( err == noErr )
    {
        Str255 alert_text;

        alert_text[0] = 0;
        do_p_strcat ( alert_text, "\pDo you really want to replace \"" );
        do_p_strcat ( alert_text, file_name );
        do_p_strcat ( alert_text, "\"\p?" );

        item_hit = do_Two_Button_Alert ( kAlertCautionAlert, alert_text, "\pThis action cannot be undone.",
"\pReplace", "\pCancel" );

        // If the user chose to replace it, we need to try to open it and see if it is a profile.
        // If it is a profile, we can just work with its innards.
        // If it isn't a profile, we need to delete the file and create a new profile.
        if( item_hit == kAlertStdAlertOKButton )
        {
            err = CMOpenProfile ( &prof_ref, &prof_loc );

            // If there was an error, the file was not a valid profile.
            // We need to delete the file and create a new profile.
            if( err != noErr )
            {
                err = FSDelete ( &file_spec );
                if( err == noErr )
                {
                    err = CMNewProfile ( &prof_ref, &prof_loc );
                    if( err != noErr )
                        goto bail;
                }
            }
        }
        else // item_hit == kAlertStdAlertCancelButton
        {
            do_One_Button_Alert ( kAlertNoteAlert, "\pYou canceled.", NULL, "\pOK" );
            goto bail;
        }
    }
}
```

```
    }  
    // If there is a 'file not found' error, we need to create a new profile.  
    else // err == fnfErr  
    {  
        err = CMNewProfile ( &prof_ref, &prof_loc );  
        if( err != noErr )  
            goto bail;  
    }  
else // Some other error occurred that we weren't prepared for.  
{  
    goto bail;  
}  
  
// Profile Header  
// -----  
// We first look for the header that should be part of the new empty profile created above.  
err = CMGetProfileHeader ( prof_ref, (CMAppleProfileHeader *)&header );  
if( noErr == err )  
{  
    // Now determine if it's a type 1 or type 2 header.  
    // The major version rev is one byte long and comes after the size and CMType.  
    version_offset = sizeof(unsigned long) + sizeof(OSType);  
    major_rev = *((unsigned char *)&header + version_offset);  
    major_rev = do_BCD_To_Decimal(major_rev);  
  
    GetTime( &date_time );  
  
    // If the profile is a new one, CS 3.0 doesn't fill out the version member.  
    // If it's a version 2 profile, we just overwrite the version number for now.  
    // If it's not one of these cases, it could be a newer profile which we don't  
    // know anything about so we shouldn't mess with it.  
    if( major_rev == 0 || major_rev == 2 )  
    {  
        // CMNewProfile() fills in the size and profileVersion fields for us.  
        header.CMMType = kDefaultCMMSignature;  
        // In v3.0.0, this member is not filled out correctly.  
        header.profileVersion = 0x02000000;  
        header.profileClass = cmDisplayClass;  
        header.dataColorSpace = cmRGBData;  
        header.profileConnectionSpace = cmXYZData;  
        header.dateTime.year = date_time.year;  
        header.dateTime.month = date_time.month;  
        header.dateTime.dayOfTheMonth = date_time.day;  
        header.dateTime.hours = date_time.hour;  
        header.dateTime.minutes = date_time.minute;  
        header.dateTime.seconds = date_time.second;  
        header.CS2profileSignature = 'acsp';  
        header.platform = cmMacintosh;  
        header.flags = 0L;  
        header.deviceManufacturer = 0L;  
        header.deviceModel = 0L;  
        header.deviceAttributes[0] = 0L;  
        header.deviceAttributes[1] = 0L;  
        header.renderingIntent = cmPerceptual;  
        header.white.X = 0x0000f6d6;    // D50 must be specified as the illuminant in the header  
        header.white.Y = 0x00010000;    // for now. See Poynton's  
        header.white.Z = 0x0000d32d;  
        header.creator = kIccManufacturerTag;  
  
        // Set the header info.  
        err = CMSetProfileHeader ( prof_ref, (CMAppleProfileHeader *)&header );  
        if( err != noErr )  
            goto bail;  
    }  
else  
{  
    do_One_Button_Alert ( kAlertStopAlert,  
        "\pThe specified profile cannot be replaced because it is not a version 2 ICC profile.",  
        "\pPlease delete the profile manually or create a new profile with a different name.",  
        "\pOK" );  
    goto bail;  
}  
}  
else  
{  
    goto bail;  
}  
  
// Description Tag  
// -----  
// We first look for an existing description tag and delete it if it exists.  
// Then we create and write our custom tag.  
err = CMProfileElementExists ( prof_ref, cmSigProfileDescriptionType, &found_it );  
if( noErr == err )  
{  
    if( found_it )
```

```
{
    err = CMRemoveProfileElement ( prof_ref, cmSigProfileDescriptionType );
    if( err != noErr )
        goto bail;
}

// Set the internal name.
if( colorsync_name[0] == 0 )
    err = do_Set_Profile_Description_Tag ( prof_ref, file_name, 0);
else
    err = do_Set_Profile_Description_Tag ( prof_ref, colorsync_name, 0);

if( err != noErr )
    goto bail;
}
else
{
    goto bail;
}

// Copyright Tag
// -----
// We first look for an existing copyright tag and delete it if it exists.
// Then we create and set our custom tag.
err = CMPProfileElementExists ( prof_ref, cmCopyrightTag, &found_it );
if( noErr == err )
{
    if( found_it )
    {
        err = CMRemoveProfileElement ( prof_ref, cmCopyrightTag );
        if( err != noErr )
            goto bail;
    }

    err = do_Set_Profile_Copyright_Tag ( prof_ref, "\pCopyright 1998-2001 bergdesign inc.", 0 );
    if( err != noErr )
        goto bail;
}
else
{
    goto bail;
}

// redColorantTag
// -----
err = do_Get_Profile_Colorant_Tag( prof_ref, cmRedColorantTag, &red_color );
if( noErr == err || cmElementTagNotFound == err )
{
    if( work ) // User selection
    {
        red_color.X = globals->tri_data[globals->tri_choice].values->red_X;
        red_color.Y = globals->tri_data[globals->tri_choice].values->red_Y;
        red_color.Z = globals->tri_data[globals->tri_choice].values->red_Z;
    }
    else // sRGB Profile
    {
        red_color.X = FloatToFixed(0.43604); // 0x00006fa0
        red_color.Y = FloatToFixed(0.22249); // 0x000038f5
        red_color.Z = FloatToFixed(0.01393); // 0x00000391
    }

    err = do_Set_Profile_Colorant_Tag( prof_ref, cmRedColorantTag, &red_color );
    if( err )
        goto bail;
}
else
{
    goto bail;
}

// greenColorantTag
// -----
err = do_Get_Profile_Colorant_Tag( prof_ref, cmGreenColorantTag, &green_color );
if( noErr == err || cmElementTagNotFound == err )
{
    if( work ) // User selection
    {
        green_color.X = globals->tri_data[globals->tri_choice].values->green_X;
        green_color.Y = globals->tri_data[globals->tri_choice].values->green_Y;
        green_color.Z = globals->tri_data[globals->tri_choice].values->green_Z;
    }
    else // sRGB Profile
    {
        green_color.X = FloatToFixed(0.38512); // 0x00006297
        green_color.Y = FloatToFixed(0.71690); // 0x0000b787
        green_color.Z = FloatToFixed(0.09708); // 0x000018da
    }
}
```

```
err = do_Set_Profile_Colorant_Tag( prof_ref, cmGreenColorantTag, &green_color );
if( err )
    goto bail;
}
else
{
    goto bail;
}

// blueColorantTag
// -----
err = do_Get_Profile_Colorant_Tag( prof_ref, cmBlueColorantTag, &blue_color );
if( noErr == err || cmElementTagNotFound == err )
{
    if( work ) // User selection
    {
        blue_color.X = globals->tri_data[globals->tri_choice].values->blue_X;
        blue_color.Y = globals->tri_data[globals->tri_choice].values->blue_Y;
        blue_color.Z = globals->tri_data[globals->tri_choice].values->blue_Z;
    }
    else // sRGB Profile
    {
        blue_color.X = FloatToFixed(0.14305); // 0x0000249f
        blue_color.Y = FloatToFixed(0.06061); // 0x00000f84
        blue_color.Z = FloatToFixed(0.71391); // 0x0000b6c3
    }

    err = do_Set_Profile_Colorant_Tag( prof_ref, cmBlueColorantTag, &blue_color );
    if( err )
        goto bail;
}
else
{
    goto bail;
}

// mediaWhitePointTag
// -----
err = do_Get_Profile_Colorant_Tag( prof_ref, cmMediaWhitePointTag, &white_color );
if( noErr == err || cmElementTagNotFound == err )
{
    if( work ) // User selection
    {
        if( globals->tri_data[globals->tri_choice].values->native_white == 0 ) // 5000
        {
            white_color.X = globals->tri_data[globals->tri_choice].values->white_5000_X;
            white_color.Y = globals->tri_data[globals->tri_choice].values->white_5000_Y;
            white_color.Z = globals->tri_data[globals->tri_choice].values->white_5000_Z;
        }
        else if ( globals->tri_data[globals->tri_choice].values->native_white == 1 ) // 6500
        {
            white_color.X = globals->tri_data[globals->tri_choice].values->white_6500_X;
            white_color.Y = globals->tri_data[globals->tri_choice].values->white_6500_Y;
            white_color.Z = globals->tri_data[globals->tri_choice].values->white_6500_Z;
        }
        else // 9300
        {
            white_color.X = globals->tri_data[globals->tri_choice].values->white_9000_X;
            white_color.Y = globals->tri_data[globals->tri_choice].values->white_9000_Y;
            white_color.Z = globals->tri_data[globals->tri_choice].values->white_9000_Z;
        }
    }
    else // sRGB Profile
    {
        white_color.X = FloatToFixed(0.95045); // 0x0000f351
        white_color.Y = FloatToFixed(1.00000); // 0x00010000
        white_color.Z = FloatToFixed(1.08905); // 0x000116cc
    }

    err = do_Set_Profile_Colorant_Tag( prof_ref, cmMediaWhitePointTag, &white_color );
    if( err )
        goto bail;
}
else
{
    goto bail;
}

// mediaBlackPointTag
// -----
err = do_Get_Profile_Colorant_Tag( prof_ref, cmMediaBlackPointTag, &black_color );
if( noErr == err || cmElementTagNotFound == err )
{
    black_color.X = FloatToFixed(0.0); // 0x00000000
    black_color.Y = FloatToFixed(0.0); // 0x00000000
    black_color.Z = FloatToFixed(0.0); // 0x00000000

    err = do_Set_Profile_Colorant_Tag( prof_ref, cmMediaBlackPointTag, &black_color );
}
```

```
        if( err )
            goto bail;
    }
    else
    {
        goto bail;
    }

    // redTRCTag
    // -----
    err = do_Set_Profile_TRC_Tag( prof_ref, cmRedTRCTag, globals->target_gamma );
    if( err )
        goto bail;

    // greenTRCTag
    // -----
    err = do_Set_Profile_TRC_Tag( prof_ref, cmGreenTRCTag, globals->target_gamma );
    if( err )
        goto bail;

    // blueTRCTag
    // -----
    err = do_Set_Profile_TRC_Tag( prof_ref, cmBlueTRCTag, globals->target_gamma );
    if( err )
        goto bail;

    // Make and Model Tag ('mmod')
    // -----
    // We've added an option to pull the make and model tags so we can work around
    // the multiple part manufacturer issue that affected us on the Pismos.
    err = CMPProfileElementExists ( prof_ref, cmSigMakeAndModelType, &found_it );
    if( noErr == err )
    {
        if( found_it )
        {
            err = CMRemoveProfileElement ( prof_ref, cmSigMakeAndModelType );
            if(err)
                goto bail;
        }
    }
    else
    {
        goto bail;
    }

    // Video Card Gamma Tag
    // -----
    // We first look for an existing vcgamma tag and delete it if it exists.
    err = CMPProfileElementExists ( prof_ref, cmVideoCardGammaTag, &found_it );
    if( noErr == err )
    {
        if( found_it )
        {
            err = CMRemoveProfileElement ( prof_ref, cmVideoCardGammaTag );
            if(err)
                goto bail;
        }
    }
    else
    {
        goto bail;
    }

    err = do_Set_Profile_VCGamma_Tag ( prof_ref );
    if(err)
        goto bail;

    // Custom measurement data Tag
    // -----
    // We first look for an existing tag and delete it if it exists.

    err = CMPProfileElementExists ( prof_ref, kIccPrivateTag, &found_it );
    if( noErr == err )
    {
        if( found_it )
        {
            err = CMRemoveProfileElement ( prof_ref, kIccPrivateTag );
            if(err)
                goto bail;
        }
    }
    else
    {
        goto bail;
    }

    err = do_Set_Profile_Custom_Measurements_Tag( prof_ref );
    if( err )
```

```
goto bail;

// -----
// Save off changes to the profile and close it.
err = CMUpdateProfile ( prof_ref );
if( err )
    goto bail;

// We'll close the profile in the bail routines.

// Need to return the profile location in the globals
// so the monitors control panel knows where it is.
globals->chosen_profile_loc = prof_loc;

bail:

if( err )
{
    Str255    error_text;

    error_text[0] = 0;
    do_p_strcat ( error_text, "\pAn error of type " );
    do_p_strerrcat ( error_text, err );
    do_p_strcat ( error_text, "\p occurred while creating the profile." );
    do_One_Button_Alert( kAlertStopAlert, error_text, NULL, "\pOK" );
}

if( prof_ref != NULL )
    CMCloseProfile ( prof_ref );

return( err );
}

//
CMError o_save_profile_pane::do_Set_Profile_Description_Tag ( CMPProfileRef prof, Str255 name, ScriptCode code )
{
    CMError    err = noErr;
    Ptr        tag = NULL;
    UInt32     offset = 0;
    UInt32     nameLen = 0;

    nameLen = name[0] + 1;
    tag = NewPtrClear(23 + 2 * nameLen);
    if( tag != NULL )
    {
        // OSType tag.typeDescriptor
        *((OSType *) (tag + offset)) = cmSigProfileDescriptionType;
        offset += 4;

        // UInt32 tag.reserved
        offset += 4;

        // UInt32 tag.ASCIICount
        *((UInt32 *) (tag + offset)) = nameLen;
        offset += 4;

        // tag.ASCIIName
        do_p2c_strcpy ( tag + offset, name );
        offset += nameLen;

        // tag.UniCodeCode, UniCodeCount, UniCodeName
        offset += 8;

        // tag.ScriptCodeCode
        *((SInt16 *) (tag + offset)) = code;
        offset += 2;

        // tag.ScriptCodeCount
        *((UInt8 *) (tag + offset)) = nameLen;
        offset += 1;

        // tag.ScriptCodeName
        do_p2c_strcpy ( tag + offset, name );

        err = CMSetProfileElement ( prof, cmSigProfileDescriptionType, GetPtrSize(tag), tag );

        DisposePtr( tag );
    }
    else
    {
        err = memFullErr;
    }

    return( err );
}

//
```



```
CMError o_save_profile_pane::do_Set_Profile_Copyright_Tag ( CMPProfileRef prof, const Str255 text, ScriptCode code
)
{
    CMError      err = noErr;
    Ptr          tag = NULL;
    UInt32       offset = 0;
    UInt32       textLen = 0;

    textLen = text[0] + 1;
    tag = NewPtrClear(8 + textLen);
    if( tag != NULL )
    {
        // OSType tag.typeDescriptor
        *((OSType *)(tag + offset)) = cmSigTextType;
        offset += 4;

        // UInt32 tag.reserved
        *(tag + offset) = 0x00000000;
        offset += 4;

        // tag.ASCIIName
        do_p2c_strcpy ( tag + offset, text );

        err = CMSetProfileElement ( prof, cmCopyrightTag, GetPtrSize(tag), tag );

        DisposePtr( tag );
    }
    else
    {
        err = memFullErr;
    }

    return( err );
}

//
CMError o_save_profile_pane::do_Get_Profile_Colorant_Tag ( CMPProfileRef prof, OSType tagType, CMFixedXYZColor *xyz
)
{
    CMError      err;
    UInt32       tagSz = sizeof(CMFixedXYZColor);

    err = CMGetPartialProfileElement(prof, tagType, 8, &tagSz, (void *)xyz);
    if( err != NULL && err != cmElementTagNotFound )
    {
        unsigned char tagStr[] = "\p'xxxx'";

        *((OSType*)(tagStr+2)) = tagType; // Shortcut to turn an OSType into a pstring
        do_One_Button_Alert ( kAlertStopAlert, "\pProfile Error: Can't get colorant tag.", (const unsigned char
*)tagStr, "\pOK" );
        do_Alert_If_Error ( "\pError", err );
    }

    return( err );
}

//
CMError o_save_profile_pane::do_Set_Profile_Colorant_Tag ( CMPProfileRef prof, OSType tagType, CMFixedXYZColor *xyz
)
{
    CMError      err = noErr;
    CMXYZType    tag;

    tag.typeDescriptor = cmSigXYZType;
    tag.reserved = 0x00000000;

    if( xyz != NULL )
    {
        tag.XYZ[0] = *xyz;
    }
    else
    {
        tag.XYZ[0].X = FloatToFixed(0.333333);
        tag.XYZ[0].Y = FloatToFixed(0.333333);
        tag.XYZ[0].Z = FloatToFixed(0.333333);
    }
    err = CMSetProfileElement( prof, tagType, sizeof(tag), &tag );
    if(err)
    {
        unsigned char tagStr[] = "\p'xxxx'";

        *((OSType*)(tagStr+2)) = tagType;
        do_One_Button_Alert ( kAlertStopAlert, "\pProfile Error: Can't set colorant tag.", (const unsigned char
*)tagStr, "\pOK" );
    }
}
```

```
    return( err );
}

//
CMError o_save_profile_pane::do_Set_Profile_TRC_Tag( CMPProfileRef prof, OSType tagType, float gamma )
{
    CMError      err = noErr;
    CMCurveType  curve;

    curve.typeDescriptor = cmSigCurveType;
    curve.reserved = 0x00000000;
    curve.countValue = 1;
    // curve.data[0] = FloatToFixed(gamma);

    // The count value specifies the number of entries in the curve table except as follows:
    // When count is 0, then a linear response (slope equal to 1.0) is assumed.
    // When count is 1, then the data entry is interpreted as a simple gamma value encoded as a u8Fixed8Number.
    // Gamma is interpreted canonically and not as an inverse.
    // u8Fixed8Number: This type represents a fixed unsigned 2 byte/16 bit quantity which has 8 fractional bits.
    // An example of this encoding is:
    // 0          0000h
    // 1.0        0100h
    // 255 + (255/256) FFFFh
    // Otherwise, the 16-bit unsigned integers in the range 0 to 65535
    // linearly map to curve values in the interval [0.0, 1.0].

    // A quick method to convert to the u8Fixed8Number format. As long as we pass
    // a reasonable gamma value in here, we don't have to worry about unexpected values
    // that result from the cast to an unsigned short.
    if( gamma != 0 )
        curve.data[0] = FloatToUShortFixed( gamma );
    else
        curve.data[0] = FloatToUShortFixed( 1.8 );

    err = CMSetProfileElement( prof, tagType, sizeof(CMCurveType), (void *)&curve );
    if( err != noErr )
    {
        unsigned char tagStr[] = "\p'xxxx'";

        (*(OSType*)(tagStr+2)) = tagType;
        do_One_Button_Alert ( kAlertStopAlert, "\pProfile Error: Can't set TRC tag.", (const unsigned char *)tagStr
"\pOK" );
    }

    return( err );
}

//
CMError o_save_profile_pane::do_Set_Profile_VCGamma_Tag ( CMPProfileRef prof_ref )
{
    CMError      err = noErr;
    unsigned short num_channels = 3; // common default
    unsigned short entry_count = 256; // common default
    unsigned short entry_size = 1; // common default
    // the CMVideoCardGammaType has the 'vcgt' tag in front of it for the profile
    // a CMVideoCardGamma is the gamma struct itself
    CMVideoCardGammaType *vc_gamma_type = NULL;
    UInt32 vc_gamma_size = 0;
    long size_diff = 0;

    // Then we build a new video card gamma table.
    if( globals->this_component.this_dev_info->channel_count > 1 )
        num_channels = 3;
    else
        num_channels = 1;

    entry_count = globals->this_component.this_dev_info->entry_count;
    entry_size = globals->this_component.this_dev_info->entry_size;

    // Since the structures can be variable in length, we need to set
    // memory aside before we start. This amount will insure that it
    // will be large enough for anything that we can put in it,
    // plus some due to the difference in member sizes in the unions.
    // The "size_diff" removes the extra space that results from the difference
    // in size between a gamma table and a formula in the CMVideoCardGamma union.

    // size_diff = ( sizeof(CMVideoCardGammaTable) - sizeof(char) ) - sizeof(CMVideoCardGammaFormula);
    size_diff = ( sizeof(UInt16) + sizeof(UInt16) + sizeof(UInt16) ) - sizeof(CMVideoCardGammaFormula);
    vc_gamma_size = sizeof( CMVideoCardGammaType ) + ( num_channels * entry_count * entry_size ) + size_diff;

    // DEBUG_VAR_PRINT( "sizeof(CMVideoCardGammaTable) = %d", sizeof(CMVideoCardGammaTable) );
    // DEBUG_VAR_PRINT( "sizeof(CMVideoCardGammaFormula) = %d", sizeof(CMVideoCardGammaFormula) );
    // DEBUG_VAR_PRINT( "sizeof(CMVideoCardGamma) = %d", sizeof(CMVideoCardGamma) );
    // DEBUG_VAR_PRINT( "sizeof(CMVideoCardGammaType) = %d", sizeof(CMVideoCardGammaType) );
    // DEBUG_VAR_PRINT( "vc_gamma_size = %d", vc_gamma_size );
}
```

```
vc_gamma_type = (CMVideoCardGammaType *)calloc( 1, vc_gamma_size );
if( !vc_gamma_type )
    goto bail;

// First the fields of the CMVideoCardGammaType struct.
// This is the tag for the profile.
vc_gamma_type->typeDescriptor = cmSigVideoCardGammaType;
vc_gamma_type->reserved = 0L;

    control_points_to_table( &(vc_gamma_type->gamma), globals->target_perceptual, globals->target_gamma,
&(globals->this_component) );

// Finally, we add our new video card gamma tag to the profile.
err = CMSetProfileElement ( prof_ref, cmVideoCardGammaTag, vc_gamma_size, vc_gamma_type );
if( err )
    goto bail;

bail:

    if( vc_gamma_type != NULL )
        free( vc_gamma_type );

    return( err );
}

//
CMError o_save_profile_pane::do_Set_Profile_Custom_Measurements_Tag ( CMPProfileRef prof_ref )
{
    CMError      err = noErr;
    //UInt32      cp_info_size = 0;
    char         *data = NULL;
    char         *offset = NULL;
    UInt32       data_size = 0;
    short        mbar_height = GetMBarHeight();
    GDHandle     device = NULL;

    // We build our custom measurement data
    UInt32 cp_info_size_r = get_control_point_info_size( globals->this_component.cp_r->count );
    UInt32 cp_info_size_g = get_control_point_info_size( globals->this_component.cp_g->count );
    UInt32 cp_info_size_b = get_control_point_info_size( globals->this_component.cp_b->count );

    // 16384 bytes is bigger than we need, but safe
    data_size = 16384 + cp_info_size_r + cp_info_size_g + cp_info_size_b;

    data = (char *)calloc( 1, data_size );
    if( NULL == data )
        goto bail;

    offset = data;

    // OSType tag.typeDescriptor
    *((OSType *)offset) = kIccPrivateTag;
    offset += A_LONG;

    // OSType tag.reserved
    *((OSType *)offset) = 0x00000000;
    offset += A_LONG;

    // The tag length indicates the length of the data to follow in bytes,
    // not including the tag identifier or tag length itself.

    //
    // indicates monochromatic or trichromatic (expert) mode
    // might use globals->expert_mode in some way too?
    *((UInt32 *)offset) = 'chan';
    offset += A_LONG;
    *((UInt32 *)offset) = A_LONG;
    offset += A_LONG;
    *((SInt32 *)offset) = globals->number_of_channels;
    offset += A_LONG;

    //
    // origin of profile - new or existing
    *((UInt32 *)offset) = 'orig';
    offset += A_LONG;
    *((UInt32 *)offset) = A_LONG;
    offset += A_LONG;
    if( globals->chosen_profile_index == -1 )    // -1 = new, 0 = existing
    {
        *((SInt32 *)offset) = -1;
    }
    else
    {
        *((SInt32 *)offset) = 0;
    }
    offset += A_LONG;
}
```

```
//
// display type
*((UInt32 *)offset) = 'dstp';
offset += A_LONG;
*((UInt32 *)offset) = 4;
offset += A_LONG;
*((SInt32 *)offset) = globals->display_type;    // type
offset += A_LONG;

//
// control types
*((UInt32 *)offset) = 'ctrl';
offset += A_LONG;
if( globals->controls_type == kDisplayControlsNone )
{
    *((UInt32 *)offset) = 4;    // data size
    offset += A_LONG;
    *((UInt32 *)offset) = 0;    // quantity
    offset += A_LONG;
}
else if( globals->controls_type == kDisplayControlsBrightnessOnly )
{
    *((UInt32 *)offset) = 12;    // data size
    offset += A_LONG;
    *((UInt32 *)offset) = 1;    // quantity
    offset += A_LONG;
    *((SInt32 *)offset) = kDisplayControlBlackLevel;
    offset += A_LONG;
    *((SInt32 *)offset) = 100;    // level, 0 to 100
    offset += A_LONG;
}
else if( globals->controls_type == kDisplayControlsContrastOnly )
{
    *((UInt32 *)offset) = 12;    // data size
    offset += A_LONG;
    *((UInt32 *)offset) = 1;    // quantity
    offset += A_LONG;
    *((SInt32 *)offset) = kDisplayControlPicture;
    offset += A_LONG;
    *((SInt32 *)offset) = 100;    // level, 0 to 100
    offset += A_LONG;
}
else if( globals->controls_type == kDisplayControlsBrightnessAndContrast )
{
    *((UInt32 *)offset) = 20;    // data size
    offset += A_LONG;
    *((UInt32 *)offset) = 2;    // quantity
    offset += A_LONG;
    *((SInt32 *)offset) = kDisplayControlBlackLevel;
    offset += A_LONG;
    *((SInt32 *)offset) = 100;    // level, 0 to 100
    offset += A_LONG;
    *((SInt32 *)offset) = kDisplayControlPicture;
    offset += A_LONG;
    *((SInt32 *)offset) = 100;    // level, 0 to 100
    offset += A_LONG;
}

//
// display resolution
Point resolution = do_Get_Video_Resolution_By_DisplayID( globals->display_id );

*((UInt32 *)offset) = 'reso';
offset += A_LONG;
*((UInt32 *)offset) = 8;
offset += A_LONG;
*((SInt32 *)offset) = resolution.h; // pixels
offset += A_LONG;
*((SInt32 *)offset) = resolution.v; // pixels
offset += A_LONG;

//
// refresh rate/frequency
int frequency = do_Get_Main_Video_Frequency();

*((UInt32 *)offset) = 'freq';
offset += A_LONG;
*((UInt32 *)offset) = 4;
offset += A_LONG;
*((SInt32 *)offset) = frequency;
offset += A_LONG;

//
// bit depth
```

```
int depth = do_Get_Video_Depth_By_DisplayID( globals->display_id );

*((UInt32 *)offset) = 'bitd';
offset += A_LONG;
*((UInt32 *)offset) = 8;
offset += A_LONG;
if( depth > 1 ) // 0 = monochrome, 1 = color
    *((SInt32 *)offset) = 1;
else
    *((SInt32 *)offset) = 0;
offset += A_LONG;
*((SInt32 *)offset) = depth;    // bit depth
offset += A_LONG;

//
// date and time profile was created
struct tm    now_tm;
time_t      now_t;
now_t = time( NULL );
now_tm = localtime( &now_t );
*((UInt32 *)offset) = 'date';
offset += A_LONG;
*((UInt32 *)offset) = sizeof(struct tm);
offset += A_LONG;
*((struct tm *)offset) = *now_tm;    // data
offset += sizeof(struct tm);

/*
//
// EDID block
*((UInt32 *)offset) = 'edid';
offset += A_LONG;
*((UInt32 *)offset) = 128;
offset += A_LONG;
*((SInt32 *)offset) = 0;    // EDID data
offset += 128;

*/
//
// target gamma
*((UInt32 *)offset) = 'gama';
offset += A_LONG;
*((UInt32 *)offset) = A_FLOAT;
offset += A_LONG;
if( globals->target_perceptual == true )
{
    *((float *)offset) = -1.0;    // -1 = perceptual
}
else
{
    *((float *)offset) = globals->target_gamma;    // -1 = perceptual
}
offset += A_LONG;

//
// resolution independent measurement data
*((UInt32 *)offset) = 'rimd';
offset += A_LONG;
*((UInt32 *)offset) = 4 + cp_info_size_r + cp_info_size_g + cp_info_size_b;
offset += A_LONG;
*((UInt32 *)offset) = 0x00000100;    // version
offset += A_LONG;
err = stream_out_control_point_info( offset, globals->this_component.cp_r );
offset += cp_info_size_r;
err = stream_out_control_point_info( offset, globals->this_component.cp_g );
offset += cp_info_size_g;
err = stream_out_control_point_info( offset, globals->this_component.cp_b );
offset += cp_info_size_b;

data_size = offset - data;

// Finally, we add our new custom measurement data tag to the profile.
err = CMSetProfileElement ( prof_ref, kIccPrivateTag, data_size, data );
if( err )
    goto bail;

bail:

if( data != NULL )
    free( data );

return( err );

}

//
pascal ControlKeyFilterResult o_save_profile_pane::do_Edit_Text_Field_Filter ( ControlHandle control, short
*keyCode, short *charCode, short *modifiers )
{
#pragma unused ( control, keyCode, modifiers )
```

```
Str255          the_string;
o_save_profile_pane *this_class;
ControlHandle    file_name_field = NULL;
ControlKeyFilterResult allow_key = kControlKeyFilterPassKey;

    this_class = (o_save_profile_pane*)GetControlReference( control );

    GetDialogItemAsControl ( this_class->window_ref, kSaveProfilePaneEditTextFileName + this_class->num_orig_items,
&file_name_field );

    switch ( *charCode )
    {
        // No matter which field it is, some keys get trapped.
        case kNullCharCode:
        case kHomeCharCode:
        case kEnterCharCode:
        case kEndCharCode:
        case kBellCharCode:
        case kTabCharCode:
        case kLineFeedCharCode:
        case kPageUpCharCode:
        case kPageDownCharCode:
        case kReturnCharCode:
        case kEscapeCharCode:
        case kNonBreakingSpaceCharCode:
        {
            SysBeep(1);
            allow_key = kControlKeyFilterBlockKey;
            break;
        }
        default:
        {
            // If the current field is the file name field, we need to limit the file name length to 31 chars.
            if( control == file_name_field )
            {
                do_Get_Text_Of_DItem_As_PString ( this_class->window_ref, kSaveProfilePaneEditTextFileName +
this_class->num_orig_items, the_string );

                // If the name length is already 31 characters, we need to block any keystrokes that would add char
                if( the_string[0] >= 31 )
                {
                    switch ( *charCode )
                    {
                        // We pass thru editing keys...
                        case kBackspaceCharCode:
                        case kDeleteCharCode:
                        case kClearCharCode:
                        case kHomeCharCode:
                        case kEndCharCode:
                        case kLeftArrowCharCode:
                        case kRightArrowCharCode:
                        case kUpArrowCharCode:
                        case kDownArrowCharCode:
                        {
                            allow_key = kControlKeyFilterPassKey;
                            break;
                        }
                        // ...but block anything else.
                        default:
                        {
                            SysBeep(1);
                            allow_key = kControlKeyFilterBlockKey;
                            break;
                        }
                    }
                }
            }
        }
    }
    else
    {
        allow_key = kControlKeyFilterPassKey;
        break;
    }
}

return ( allow_key );
}
```

```
//  
//  
//  
#ifndef __o_cal_slider_  
#define __o_cal_slider_  
  
#ifdef __APPLE_CC_  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Appearance.h>  
#include <Controls.h>  
#include <MacWindows.h>  
#include <Events.h>  
#endif  
#endif  
#endif  
  
#include "my_macros.h"  
#include "my_quickdraw.h"  
  
// These names represent dimensions for horizontal orientation  
enum  
{  
    kMySliderIndicatorWidth = 11,  
    kMySliderIndicatorHeight = 12,  
    kMySliderButtonWidth = 16,  
    kMySliderButtonHeight = 16,  
    kMySliderTrackHeight = 16  
};  
  
enum  
{  
    kMySliderUpDownButtonDelayTicks = 10  
};  
  
//  
  
class o_cal_slider  
{  
private:  
  
protected:  
  
    ControlHandle      slider;  
    ControlUserPaneDrawUPP slider_draw_proc;  
    ControlUserPaneHitTestUPP slider_hit_test_proc;  
    ControlUserPaneActivateUPP slider_activate_proc;  
    ControlUserPaneTrackingUPP slider_tracking_proc;  
  
    Boolean            slider_is_vertical;  
    Rect               indicator_rect;  
    Rect               track_rect;  
    short              track_start, track_stop, track_range;  
    Rect               up_button_rect;  
    Rect               down_button_rect;  
    Rect               page_up_rect;  
    Rect               page_down_rect;  
    Rect               legend_rect;  
  
    static void        do_Slider_Draw_Proc ( ControlHandle, short );  
    static ControlPartCode do_Slider_Hit_Test_Proc ( ControlHandle, Point );  
    static void        do_Slider_Activate_Proc ( ControlHandle, Boolean );  
    static ControlPartCode do_Slider_Tracking_Proc ( ControlHandle, Point, ControlActionUPP );  
    void               do_Calc_Control_Metrics ();  
    void               do_Calc_Control_Value_From_Indicator ( short );  
    void               do_Calc_Indicator_From_Control_Value ();  
  
public:  
  
    // constructors & destructors  
    o_cal_slider ( WindowPtr, Rect *, ControlHandle );  
    ~o_cal_slider ();  
  
    Boolean            do_Handle_Click ( EventRecord * );  
};  
  
#endif /* __o_cal_slider__ */
```

```
//  
//  
//  
#include "o_cal_slider.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
//  
o_cal_slider::o_cal_slider ( WindowPtr window_ptr, Rect *slider_rect, ControlHandle embedder )  
{  
    slider = NULL;  
    slider_draw_proc = NULL;  
    slider_hit_test_proc = NULL;  
    slider_activate_proc = NULL;  
    slider_tracking_proc = NULL;  
  
    slider = NewControl (    window_ptr,  
                            slider_rect,  
                            NULL,  
                            false,  
                            (short)kControlWantsActivate + kControlHandlesTracking,  
                            (short)0,  
                            (short)0,  
                            (short)kControlUserPaneProc,  
                            (long)this );  
  
    if ( slider )  
    {  
        EmbedControl ( slider, embedder );  
  
        //    slider_draw_proc = NewControlUserPaneDrawProc ( o_cal_slider::do_Slider_Draw_Proc );  
        slider_draw_proc = NewControlUserPaneDrawUPP ( (ControlUserPaneDrawProcPtr)o_cal_slider::do_Slider_Draw_Proc );  
  
        //    slider_hit_test_proc = NewControlUserPaneHitTestProc ( o_cal_slider::do_Slider_Hit_Test_Proc );  
        slider_hit_test_proc = NewControlUserPaneHitTestUPP ( (ControlUserPaneHitTestProcPtr)o_cal_slider::do_Slider_Hit_Test_Proc );  
  
        //    slider_activate_proc = NewControlUserPaneActivateProc ( o_cal_slider::do_Slider_Activate_Proc );  
  
        //    slider_tracking_proc = NewControlUserPaneTrackingProc ( o_cal_slider::do_Slider_Tracking_Proc );  
        slider_tracking_proc = NewControlUserPaneTrackingUPP ( (ControlUserPaneTrackingProcPtr)o_cal_slider::do_Slider_Tracking_Proc );  
  
        SetControlData( slider, 0, kControlUserPaneDrawProcTag, sizeof( ControlUserPaneDrawUPP ),  
(Ptr)&slider_draw_proc );  
        SetControlData( slider, 0, kControlUserPaneHitTestProcTag, sizeof( ControlUserPaneHitTestUPP ),  
(Ptr)&slider_hit_test_proc );  
        //    SetControlData( slider, 0, kControlUserPaneActivateProcTag, sizeof( ControlUserPaneDrawUPP ),  
(Ptr)&slider_activate_proc );  
        SetControlData( slider, 0, kControlUserPaneTrackingProcTag, sizeof( ControlUserPaneTrackingUPP ),  
(Ptr)&slider_tracking_proc );  
  
        if ( ( slider_rect->bottom - slider_rect->top ) > ( slider_rect->right - slider_rect->left ) )  
            slider_is_vertical = true;  
        else  
            slider_is_vertical = false;  
  
        SetControlMinimum ( slider, 0 );  
        SetControlMaximum ( slider, 10 );  
        SetControlValue ( slider, 5 );  
  
        ShowControl ( slider );  
    }  
}  
  
//  
o_cal_slider::~o_cal_slider ()  
{  
    DisposeControl ( slider );  
  
    if ( slider_draw_proc )  
    {  
        //    DisposeRoutineDescriptor( slider_draw_proc );  
        DisposeControlUserPaneDrawUPP( slider_draw_proc );  
    }  
  
    if ( slider_hit_test_proc )  
    {  
        //    DisposeRoutineDescriptor( slider_hit_test_proc );  
        DisposeControlUserPaneHitTestUPP( slider_hit_test_proc );  
    }  
}
```



```
if ( slider_activate_proc )
{
// DisposeRoutineDescriptor( slider_activate_proc );
DisposeControlUserPaneActivateUPP( slider_activate_proc );
}

if ( slider_tracking_proc )
{
// DisposeRoutineDescriptor( slider_tracking_proc );
DisposeControlUserPaneTrackingUPP( slider_tracking_proc );
}
}

//
#pragma mark -
//

Boolean o_cal_slider::do_Handle_Click ( EventRecord *event )
{
Boolean click_handled = false;
Point where;

where = event->where;
GlobalToLocal ( &where ); // the current port must be correct or GlobalToLocal won't work right

if ( TestControl ( slider, where ) != 0 )
{
HandleControlClick ( slider, where, event->modifiers, (ControlActionUPP)-1L );
click_handled = true;
}

return ( click_handled );
}

//

void o_cal_slider::do_Slider_Draw_Proc ( ControlHandle control, short part )
{
o_cal_slider *this_o;
CGrafPtr port;
GDHandle gdh;
//ColorPenState state;
ThemeDrawingState state;
WindowPtr window_ptr;
RgnHandle saved_clip_rgn = NULL;
RGBColor white, black;
Rect bounds;
Boolean active;

DEBUG_VAR_PRINT("Called do_Slider_Draw_Proc(part = %d)",part);

this_o = (o_cal_slider*)GetControlReference( control );
// window_ptr = (**control).ctrlOwner;
window_ptr = GetControlOwner( control );
if ( IsWindowCollapsed ( window_ptr ) )
return;

active = IsControlActive (control);
if ( active )
{
white.red = white.green = white.blue = 65535;
}
else
{
white.red = white.green = white.blue = 32768;
}

black.red = black.green = black.blue = 0;

// bounds = (**control).ctrlRect;
GetControlBounds( control, &bounds );
this_o->do_Calc_Control_Metrics();

GetGWorld ( &port, &gdh );
// SetGWorld ( (CGrafPtr)window_ptr, NULL );
SetPortWindowPort(window_ptr);

// GetColorAndPenState( &state );
GetThemeDrawingState( &state );

saved_clip_rgn = NewRgn();
GetClip( saved_clip_rgn );
ClipRect( &bounds );

// begin drawing
```

```
if ( part == kControlIndicatorPart || part == kControlNoPart )
{
    RGBForeColor ( &white );
    EraseRect ( &(this_o->track_rect) );
    FrameRect ( &(this_o->track_rect) );

    if ( StillDown() )
    {
        PaintRect ( &(this_o->indicator_rect) );
        RGBForeColor ( &black );
    }
    else
    {
        FrameRect ( &(this_o->indicator_rect) );
    }

    if ( this_o->slider_is_vertical )
    {
        MoveTo ( this_o->indicator_rect.left + 2, this_o->indicator_rect.top + 3 );
        LineTo ( this_o->indicator_rect.right - 3, this_o->indicator_rect.top + 3 );
        MoveTo ( this_o->indicator_rect.left + 2, this_o->indicator_rect.top + 5 );
        LineTo ( this_o->indicator_rect.right - 3, this_o->indicator_rect.top + 5 );
        MoveTo ( this_o->indicator_rect.left + 2, this_o->indicator_rect.top + 7 );
        LineTo ( this_o->indicator_rect.right - 3, this_o->indicator_rect.top + 7 );
    }
    else
    {
        MoveTo ( this_o->indicator_rect.left + 3, this_o->indicator_rect.top + 2 );
        LineTo ( this_o->indicator_rect.left + 3, this_o->indicator_rect.bottom - 3 );
        MoveTo ( this_o->indicator_rect.left + 5, this_o->indicator_rect.top + 2 );
        LineTo ( this_o->indicator_rect.left + 5, this_o->indicator_rect.bottom - 3 );
        MoveTo ( this_o->indicator_rect.left + 7, this_o->indicator_rect.top + 2 );
        LineTo ( this_o->indicator_rect.left + 7, this_o->indicator_rect.bottom - 3 );
    }
}

if ( part == kControlUpButtonPart || part == kControlNoPart )
{
    PolyHandle      arrow;

    RGBForeColor ( &white );
    EraseRect ( &(this_o->up_button_rect) );

    arrow = OpenPoly();

    if ( this_o->slider_is_vertical )
    {
        MoveTo ( this_o->up_button_rect.left + 4, this_o->up_button_rect.top + 9 );
        LineTo ( this_o->up_button_rect.right - 5, this_o->up_button_rect.top + 9 );
        LineTo ( this_o->up_button_rect.right - (kMySliderButtonHeight/2), this_o->up_button_rect.top + 6 );
        LineTo ( this_o->up_button_rect.right - (kMySliderButtonHeight/2) - 1, this_o->up_button_rect.top + 6 );
        LineTo ( this_o->up_button_rect.left + 4, this_o->up_button_rect.top + 9 );
    }
    else
    {
        MoveTo ( this_o->up_button_rect.left + 9, this_o->up_button_rect.bottom - 5 );
        LineTo ( this_o->up_button_rect.left + 9, this_o->up_button_rect.top + 4 );
        LineTo ( this_o->up_button_rect.left + 6, this_o->up_button_rect.top + (kMySliderButtonHeight/2) - 1 );
        LineTo ( this_o->up_button_rect.left + 6, this_o->up_button_rect.top + (kMySliderButtonHeight/2) );
        LineTo ( this_o->up_button_rect.left + 9, this_o->up_button_rect.bottom - 5 );
    }

    ClosePoly();

    if ( StillDown() )
    {
        PaintRect ( &(this_o->up_button_rect) );
        RGBForeColor ( &black );
        FramePoly ( arrow );
        PaintPoly ( arrow );
    }
    else
    {
        FrameRect ( &(this_o->up_button_rect) );
        FramePoly ( arrow );
        PaintPoly ( arrow );
    }

    KillPoly(arrow);
}

if ( part == kControlDownButtonPart || part == kControlNoPart )
{
    PolyHandle      arrow;

    RGBForeColor ( &white );
    EraseRect ( &(this_o->down_button_rect) );
```

```
        arrow = OpenPoly();

        if ( this_o->slider_is_vertical )
        {
            MoveTo ( this_o->down_button_rect.left + 4, this_o->down_button_rect.bottom - 10 );
            LineTo ( this_o->down_button_rect.right - 5, this_o->down_button_rect.bottom - 10 );
            LineTo ( this_o->down_button_rect.right - (kMySliderButtonHeight/2), this_o->down_button_rect.bottom -
);
            LineTo ( this_o->down_button_rect.right - (kMySliderButtonHeight/2) - 1, this_o->down_button_rect.botto
- 7 );
            LineTo ( this_o->down_button_rect.left + 4, this_o->down_button_rect.bottom - 10 );
        }
        else
        {
            MoveTo ( this_o->down_button_rect.right - 10, this_o->down_button_rect.top + 4 );
            LineTo ( this_o->down_button_rect.right - 10, this_o->down_button_rect.bottom - 5 );
            LineTo ( this_o->down_button_rect.right - 7, this_o->down_button_rect.bottom - (kMySliderButtonHeight/2
);
            LineTo ( this_o->down_button_rect.right - 7, this_o->down_button_rect.bottom - (kMySliderButtonHeight/2
- 1 );
            LineTo ( this_o->down_button_rect.right - 10, this_o->down_button_rect.top + 4 );
        }

        ClosePoly();

        if ( StillDown() )
        {
            PaintRect ( &(this_o->down_button_rect) );
            RGBForeColor ( &black );
            FramePoly ( arrow );
            PaintPoly ( arrow );
        }
        else
        {
            FrameRect ( &(this_o->down_button_rect) );
            FramePoly ( arrow );
            PaintPoly ( arrow );
        }

        KillPoly(arrow);
    }

    if ( part == kControlPageUpPart || part == kControlNoPart )
    {
    }

    if ( part == kControlPageDownPart || part == kControlNoPart )
    {
    }

    if ( part == kControlNoPart )
    {
        PolyHandle      scale;

        RGBForeColor ( &white );
        scale = OpenPoly();

        if ( this_o->slider_is_vertical )
        {
            MoveTo ( this_o->track_rect.left - 4, this_o->track_rect.bottom );
            LineTo ( this_o->track_rect.left - 4, this_o->track_rect.top );
            LineTo ( this_o->track_rect.left - 16, this_o->track_rect.top );
            LineTo ( this_o->track_rect.left - 4, this_o->track_rect.bottom );
        }
        else
        {
            MoveTo ( this_o->track_rect.left, this_o->track_rect.top - 4 );
            LineTo ( this_o->track_rect.right, this_o->track_rect.top - 4 );
            LineTo ( this_o->track_rect.right, this_o->track_rect.top - 16 );
            LineTo ( this_o->track_rect.left, this_o->track_rect.top - 4 );
        }

        ClosePoly();
        PaintPoly(scale);
        KillPoly(scale);
    }

    // end drawing

    SetClip( saved_clip_rgn );
    DisposeRgn( saved_clip_rgn );

    // SetColorAndPenState( &state );
    SetThemeDrawingState( state, true );
    SetGWorld ( port, gdh );
}

//
```

```
ControlPartCode o_cal_slider::do_Slider_Hit_Test_Proc ( ControlHandle control, Point where )
{
    o_cal_slider      *this_o;
    ControlPartCode    part_code;

    this_o = (o_cal_slider*)GetControlReference( control );

    if ( PtInRect ( where, &(amp;this_o->indicator_rect) ) )
    {
        part_code = kControlIndicatorPart;
    }
    else if ( PtInRect ( where, &(amp;this_o->up_button_rect) ) )
    {
        part_code = kControlUpButtonPart;
    }
    else if ( PtInRect ( where, &(amp;this_o->down_button_rect) ) )
    {
        part_code = kControlDownButtonPart;
    }
    else if ( PtInRect ( where, &(amp;this_o->page_up_rect) ) )
    {
        part_code = kControlPageUpPart;
    }
    else if ( PtInRect ( where, &(amp;this_o->page_down_rect) ) )
    {
        part_code = kControlPageDownPart;
    }
    else
    {
        part_code = kControlNoPart;
    }

    return part_code;
}

//

void o_cal_slider::do_Slider_Activate_Proc ( ControlHandle control, Boolean activating )
{
    o_cal_slider*      this_o;

    this_o = (o_cal_slider*)GetControlReference( control );

    if ( activating )
    {
        DrawOneControl ( this_o->slider );
    }
}

//

ControlPartCode o_cal_slider::do_Slider_Tracking_Proc ( ControlHandle control, Point mouse, ControlActionUPP
actionProc )
{
    o_cal_slider*      this_o;
    ControlPartCode    part;
    short              val, min, max, mouse_indicator_offset;
    Point              old_mouse;

    old_mouse = mouse;
    this_o = (o_cal_slider*)GetControlReference( control );
    min = GetControlMinimum ( control );
    max = GetControlMaximum ( control );

    part = this_o->do_Slider_Hit_Test_Proc ( control, mouse );

    if ( this_o->slider_is_vertical )
        mouse_indicator_offset = mouse.v - this_o->indicator_rect.top;
    else
        mouse_indicator_offset = mouse.h - this_o->indicator_rect.left;

    switch ( part )
    {
        case kControlPageUpPart:
        case kControlPageDownPart:
        {
            mouse_indicator_offset = kMySliderIndicatorWidth / 2;

            if ( this_o->slider_is_vertical )
                this_o->do_Calc_Control_Value_From_Indicator ( mouse.v - mouse_indicator_offset );
            else
                this_o->do_Calc_Control_Value_From_Indicator ( mouse.h - mouse_indicator_offset );
        }
        case kControlIndicatorPart:
        {
            do_Slider_Draw_Proc ( control, part );
        }
    }
}
```

```
while ( StillDown () )
{
    GetMouse ( &mouse );

    if ( this_o->slider_is_vertical )
    {
        if ( mouse.v != old_mouse.v )
            this_o->do_Calc_Control_Value_From_Indicator ( mouse.v - mouse_indicator_offset );
    }
    else
    {
        if ( mouse.h != old_mouse.h )
            this_o->do_Calc_Control_Value_From_Indicator ( mouse.h - mouse_indicator_offset );
    }

    old_mouse = mouse;
    SystemTask();
}

do_Slider_Draw_Proc ( control, kControlNoPart );
break;
}
case kControlUpButtonPart:
case kControlDownButtonPart:
{
    Boolean            times_up = true;
    short              new_val, increment;
    long               ticks;
    ControlPartCode    part_down;

    do_Slider_Draw_Proc ( control, part );

    if ( part == kControlUpButtonPart )
        increment = -1;
    else if ( part == kControlDownButtonPart )
        increment = 1;

    ticks = TickCount();

    while ( StillDown() )
    {
        val = GetControlValue ( control );

        if ( times_up )
        {
            new_val = val + increment;
            new_val = MAX ( min, MIN ( new_val, max ) );

            if ( new_val != val )
            {
                SetControlValue ( control, new_val );
            }
        }

        GetMouse ( &mouse );
        part_down = this_o->do_Slider_Hit_Test_Proc ( control, mouse );

        if ( ( TickCount() > ( ticks + kMySliderUpDownButtonDelayTicks ) ) &&
            ( part_down == kControlUpButtonPart || part_down == kControlDownButtonPart ) )
        {
            times_up = true;
            ticks = TickCount();
        }
        else
        {
            times_up = false;
        }
    }

    SystemTask();
}

do_Slider_Draw_Proc ( control, kControlNoPart );
break;
}
default:
{
    break;
}
}

return ( part );
}

//
void o_cal_slider::do_Calc_Control_Metrics ()
```

```
{
    Rect    bounds;

    // bounds = (**slider).ctrlRect;
    GetControlBounds( slider, &bounds );

    // remember that width and height constants are reversed for vertical orientation
    if ( slider_is_vertical )
    {
        // drawing & hit-test rect
        up_button_rect.right = bounds.right;
        up_button_rect.left = up_button_rect.right - kMySliderButtonHeight;
        up_button_rect.top = bounds.top;
        up_button_rect.bottom = up_button_rect.top + kMySliderButtonWidth;

        // drawing & hit-test rect
        down_button_rect.right = up_button_rect.right;
        down_button_rect.left = up_button_rect.left;
        down_button_rect.bottom = bounds.bottom;
        down_button_rect.top = down_button_rect.bottom - kMySliderButtonWidth;

        // drawing rect
        track_rect.right = bounds.right;
        track_rect.left = track_rect.right - kMySliderTrackHeight;
        track_rect.top = up_button_rect.bottom - 1; // We want to overlap the buttons by 1 pixel
        track_rect.bottom = down_button_rect.top + 1;

        track_start = track_rect.top + 1 + 1;
        track_stop = track_rect.bottom - 1 - 1;
        track_range = track_stop - track_start - kMySliderIndicatorWidth;
    }
    else
    {
        // drawing & hit-test rect
        up_button_rect.left = bounds.left;
        up_button_rect.right = up_button_rect.left + kMySliderButtonWidth;
        up_button_rect.bottom = bounds.bottom;
        up_button_rect.top = up_button_rect.bottom - kMySliderButtonHeight;

        // drawing & hit-test rect
        down_button_rect.right = bounds.right;
        down_button_rect.left = down_button_rect.right - kMySliderButtonWidth;
        down_button_rect.bottom = up_button_rect.bottom;
        down_button_rect.top = up_button_rect.top;

        // drawing rect
        track_rect.left = up_button_rect.right - 1; // We want to overlap the buttons by 1 pixel
        track_rect.right = down_button_rect.left + 1;
        track_rect.bottom = bounds.bottom;
        track_rect.top = track_rect.bottom - kMySliderTrackHeight;

        track_start = track_rect.left + 1 + 1;
        track_stop = track_rect.right - 1 - 1;
        track_range = track_stop - track_start - kMySliderIndicatorWidth;
    }

    do_Calc_Indicator_From_Control_Value ();

    if ( slider_is_vertical )
    {
        page_up_rect.top = track_start;
        page_up_rect.bottom = indicator_rect.top - 1;
        page_up_rect.left = track_rect.left;
        page_up_rect.right = track_rect.right;

        page_down_rect.top = indicator_rect.bottom + 1;
        page_down_rect.bottom = track_stop;
        page_down_rect.left = track_rect.left;
        page_down_rect.right = track_rect.right;
    }
    else
    {
        page_up_rect.left = track_start;
        page_up_rect.right = indicator_rect.left - 1;
        page_up_rect.top = track_rect.top;
        page_up_rect.bottom = track_rect.bottom;

        page_down_rect.left = indicator_rect.right + 1;
        page_down_rect.right = track_stop;
        page_down_rect.top = track_rect.top;
        page_down_rect.bottom = track_rect.bottom;
    }
}

//
void o_cal_slider::do_Calc_Control_Value_From_Indicator ( short new_indicator )
```

```
{
short      control_range, control_offset;
short      val, new_val, min, max;

    val = GetControlValue ( slider );
    min = GetControlMinimum ( slider );
    max = GetControlMaximum ( slider );

    control_range = max - min;

    new_indicator = MAX ( track_start, MIN ( new_indicator, track_stop ) );
    control_offset = ( ( (float)control_range * (float)( new_indicator - track_start ) ) / (float)track_range )
+.5;
    new_val = control_offset + min;

    if ( new_val != val )
    {
        SetControlValue ( slider, new_val );
    }
}

//
void o_cal_slider::do_Calc_Indicator_From_Control_Value ()
{
short      control_range, control_offset;
short      val, min, max;

    val = GetControlValue ( slider );
    min = GetControlMinimum ( slider );
    max = GetControlMaximum ( slider );
    control_range = max - min;
    control_offset = val - min;

    if ( slider_is_vertical )
    {
        indicator_rect.top = track_start + ( (float)( track_range * control_offset ) / (float)control_range ) + .5;
        indicator_rect.bottom = indicator_rect.top + kMySliderIndicatorWidth;
        indicator_rect.right = track_rect.right - 2;
        indicator_rect.left = indicator_rect.right - kMySliderIndicatorHeight;
    }
    else
    {
        indicator_rect.left = track_start + ( (float)( track_range * control_offset ) / (float)control_range ) + .5;
        indicator_rect.right = indicator_rect.left + kMySliderIndicatorWidth;
        indicator_rect.bottom = track_rect.bottom - 2;
        indicator_rect.top = indicator_rect.bottom - kMySliderIndicatorHeight;
    }
}
```

```
//  
//  
//  
#ifndef __o_help_pane__  
#define __o_help_pane__  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Resources.h>  
#endif  
#endif  
  
#include "my_dialogs.h"  
#include "my_alerts.h"  
#include "my_quickdraw.h"  
  
enum  
{  
    kHelpPaneDITL          = 3100  
};  
  
enum  
{  
    kHelpPaneFrame          = 1,  
    kHelpPaneScrollBar      = 2,  
    kHelpPaneStaticText     = 3,  
    kHelpPaneStepNumber     = 4,  
    kHelpPaneStepText       = 5,  
    kHelpPaneStepPict       = 6  
};  
  
// Biggest thing to remember with this pane (and others like it) is last-in, first-out  
// when adding these classes to existing dialogs. This is due to the ShortenDITL()  
// routine always taking dialog items off the top. We can't take them out of the middle.  
  
class o_help_pane  
{  
public:  
    o_help_pane ( DialogRef, short, short, short, short, short );  
    ~o_help_pane ();  
  
    void        do_Item_Hit ( short );  
  
protected:  
  
private:  
    static void    do_Scroll_Feedback_Proc ( ControlHandle, short );  
    static void    do_Frame_Draw_Proc ( ControlHandle, short );  
    static void    do_Pict_Draw_Proc ( ControlHandle, short );  
  
    ControlActionUPP    scroll_bar_proc;  
    ControlUserPaneDrawUPP    frame_draw_proc;  
    ControlUserPaneDrawUPP    pict_draw_proc;  
    PicHandle              step_pict_handle;  
  
    DialogRef              window_ref;  
    SInt16                  num_orig_items;  
    short                   str_res_id;  
    short                   base_pict_res_id;  
  
};  
  
#endif /* __o_help_pane__ */
```



```
//
//                                     ©1998-2001 bergdesign inc.
//
#include "o_help_pane.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

//
o_help_pane::o_help_pane ( DialogRef dialog,
                           short items,
                           short append_rel_item,
                           short num_steps,
                           short str_res,
                           short pict_res )
{
    OSStatus      err = noErr;
    Handle        ditl_hndl;

    window_ref = dialog;
    num_orig_items = items;

    frame_draw_proc = NULL;
    pict_draw_proc = NULL;
    scroll_bar_proc = NULL;

    step_pict_handle = NULL;

    str_res_id = str_res;
    base_pict_res_id = pict_res;

    ditl_hndl = GetResource ( 'DITL', kHelpPaneDITL );

    if ( ditl_hndl )
    {
        DEBUG_PRINT("Got the help pane DITL resource");

        // A little error checking. This assumes that the dialog items start
        // at "1" and have no gaps in their numbering.
        if ( append_rel_item > num_orig_items )
            append_rel_item = overlayDITL;
        else
            append_rel_item = -append_rel_item;

        DEBUG_VAR_PRINT("Used append mode %d",append_rel_item);

        AppendDITL ( window_ref, ditl_hndl, append_rel_item );
        ReleaseResource ( ditl_hndl );

        frame_draw_proc = NewControlUserPaneDrawUPP ( (ControlUserPaneDrawProcPtr)o_help_pane::do_Frame_Draw_Proc );
        do_Set_Control_Ref_Of_DItem ( window_ref, kHelpPaneFrame + num_orig_items, (long)this );
        err = do_Set_Control_Draw_Proc_Of_DItem ( window_ref, kHelpPaneFrame + num_orig_items, frame_draw_proc );

        pict_draw_proc = NewControlUserPaneDrawUPP ( (ControlUserPaneDrawProcPtr)o_help_pane::do_Pict_Draw_Proc );
        do_Set_Control_Ref_Of_DItem ( window_ref, kHelpPaneStepPict + num_orig_items, (long)this );
        err = do_Set_Control_Draw_Proc_Of_DItem ( window_ref, kHelpPaneStepPict + num_orig_items, pict_draw_proc );

        scroll_bar_proc = NewControlActionUPP ( (ControlActionProcPtr)o_help_pane::do_Scroll_Feedback_Proc );
        do_Set_Control_Ref_Of_DItem ( window_ref, kHelpPaneScrollBar + num_orig_items, (long)this );
        do_Set_Control_Action_Of_DItem ( window_ref, kHelpPaneScrollBar + num_orig_items, scroll_bar_proc );

        // default scroll min, val & max should be 1
        if ( num_steps > 1 )
        {
            ControlHandle    temp_control = NULL;

            do_Set_Max_Value_Of_DItem ( window_ref, kHelpPaneScrollBar + num_orig_items, num_steps );
            err = GetDialogItemAsControl ( window_ref, kHelpPaneScrollBar + num_orig_items, &temp_control );
            if ( ( NULL != temp_control ) && ( noErr == err ) )
                SetControlViewSize( temp_control, 1 );
        }

        do_Set_Text_Style_Of_DItem ( window_ref, kHelpPaneStaticText + num_orig_items, NULL, NULL, bold, NULL );
        do_Set_Text_Style_Of_DItem ( window_ref, kHelpPaneStepNumber + num_orig_items, NULL, NULL, bold, teCenter );

        // do this to load up text and pict for the first step
        do_Item_Hit ( kHelpPaneScrollBar + num_orig_items );
    }
}

//
o_help_pane::~o_help_pane()
{
    // No need to NULL any variables since the class is going away.
}
```

```
if ( scroll_bar_proc )
//  DisposeRoutineDescriptor( scroll_bar_proc );
//  DisposeControlActionUPP( scroll_bar_proc );

if ( frame_draw_proc )
//  DisposeRoutineDescriptor( frame_draw_proc );
//  DisposeControlUserPaneDrawUPP( frame_draw_proc );

if ( pict_draw_proc )
//  DisposeRoutineDescriptor( pict_draw_proc );
//  DisposeControlUserPaneDrawUPP( pict_draw_proc );

if ( step_pict_handle )
    ReleaseResource ( (Handle)step_pict_handle );

ShortenDITL ( window_ref, ( CountDITL ( window_ref ) - num_orig_items ) );
}

//
void
o_help_pane::do_Item_Hit ( short item_hit )
{
    switch ( item_hit - num_orig_items )
    {
        DEBUG_VAR_PRINT("o_help_pane item hit = %d",item_hit - num_orig_items);

        case kHelpPaneScrollBar:
        {
            short      value;
            Str255      step_text;
            char        step_number[11];
            short      num_steps;

            do_Get_Value_Of_DItem ( window_ref, kHelpPaneScrollBar + num_orig_items, &value );

            do_Get_Max_Value_Of_DItem ( window_ref, kHelpPaneScrollBar + num_orig_items, &num_steps );
            sprintf ( step_number, "%d of %d", value, num_steps );
            do_Set_Text_Of_DItem_As_CString ( window_ref, kHelpPaneStepNumber + num_orig_items, step_number, false );
            do_Set_Text_Of_DItem_As_Int ( window_ref, kHelpPaneStepNumber + num_orig_items, (long)value, false );

            GetIndString ( step_text, str_res_id, ( value ) );
            do_Set_Text_Of_DItem_As_PString ( window_ref, kHelpPaneStepText + num_orig_items, step_text, false );

            if ( step_pict_handle != NULL )
            {
                ReleaseResource ( (Handle)step_pict_handle );
                step_pict_handle = NULL;
            }

            step_pict_handle = GetPicture ( base_pict_res_id + ( value - 1 ) );
            do_Draw_One_Control_As_DItem ( window_ref, kHelpPaneStepPict + num_orig_items );

            break;
        }
        default:
        {
            break;
        }
    }
}

//
void
o_help_pane::do_Scroll_Feedback_Proc ( ControlHandle control, short part )
{
    short      start_value, delta, min, max;
    o_help_pane *this_class;
    WindowRef  this_window;

    this_class = (o_help_pane*)GetControlReference( control );

    start_value = GetControlValue( control );
    min = GetControlMinimum( control );
    max = GetControlMaximum( control );

    delta = 0;

    switch ( part )
    {
        case kControlUpButtonPart:
            if ( start_value > min )
                delta = MAX( -1, min - start_value );
            break;

        case kControlDownButtonPart:
            if ( start_value < max )
                delta = MAX( 1, max - start_value );
            break;
    }
}
```

```
        delta = MIN( 1, max - start_value );
        break;

    case kControlPageUpPart:
        if ( start_value > min )
            delta = MAX( -1, min - start_value );
        break;

    case kControlPageDownPart:
        if ( start_value < max )
            delta = MIN( 1, max - start_value );
        break;
}

if ( delta || part == kControlIndicatorPart )
{
    if ( delta )
    {
        SetControlValue ( control, start_value + delta );
    }
    else if ( part == kControlIndicatorPart )
    {
        // nothing
    }

    this_class->do_Item_Hit ( kHelpPaneScrollBar + this_class->num_orig_items );

    this_window = GetDialogWindow(this_class->window_ref);
    RgnHandle vis_region = NewRgn();
    GetPortVisibleRegion( GetWindowPort(this_window), vis_region );

    BeginUpdate (this_window);
    UpdateControls ( this_class->window_ref, ( ( (DialogPeek)(this_class->window_ref) )->window ).port.visRgn
);
    UpdateControls( this_window, vis_region );
    EndUpdate (this_window);

    DisposeRgn(vis_region);
}

//
void
o_help_pane::do_Frame_Draw_Proc ( ControlHandle control, short part )
{
    RgnHandle          saved_clip_rgn;
    CGrafPtr           port;
    GDHandle            gdh;
    WindowPtr           window_ptr;
    Boolean             active = false;
    Rect               bounds;
    //ColorPenState      state;
    ThemeDrawingState   state;

    // window_ptr = (**control).ctrlOwner;
    window_ptr = GetControlOwner( control );
    // bounds = (**control).ctrlRect;
    GetControlBounds( control, &bounds );
    active = IsControlActive (control);

    if ( IsWindowCollapsed ( window_ptr ) )
        return;

    GetGWorld ( &port, &gdh );
    // SetGWorld ( (CGrafPtr)window_ptr, NULL );
    SetPortWindowPort(window_ptr);
    // GetColorAndPenState( &state );
    GetThemeDrawingState( &state );

    saved_clip_rgn = NewRgn();
    GetClip( saved_clip_rgn );
    ClipRect( &bounds );

    PenNormal();

    if ( active )
    {
        do_Set_Pen ( PlatinumScrollBarActive );
    }
    else
    {
        do_Set_Pen ( PlatinumScrollBarInactive );
    }

    FrameRect ( &bounds );

    SetClip( saved_clip_rgn );
}
```

```
        DisposeRgn( saved_clip_rgn );

// SetColorAndPenState( &state );
// SetThemeDrawingState( state, true );
// SetGWorld ( port, gdh );
}

//

void
o_help_pane::do_Pict_Draw_Proc ( ControlHandle control, short part )
{
    RgnHandle          saved_clip_rgn;
    CGrafPtr           port;
    GDHandle           gdh;
    WindowPtr          window_ptr;
    Boolean             active = false;
    Rect              bounds;
    //ColorPenState     state;
    ThemeDrawingState  state;
    o_help_pane        *this_class;

    this_class = (o_help_pane *)GetControlReference( control );

    // window_ptr = (**control).ctrlOwner;
    // window_ptr = GetControlOwner( control );
    // bounds = (**control).ctrlRect;
    // GetControlBounds( control, &bounds );
    // active = IsControlActive (control);

    if ( IsWindowCollapsed ( window_ptr ) )
        return;

    GetGWorld ( &port, &gdh );
    // SetGWorld ( (CGrafPtr>window_ptr, NULL );
    SetPortWindowPort(window_ptr);
    // GetColorAndPenState ( &state );
    // GetThemeDrawingState( &state );

    saved_clip_rgn = NewRgn();
    GetClip( saved_clip_rgn );
    ClipRect( &bounds );

    NormalizeColorAndPen();

    if ( this_class->step_pict_handle != NULL )
        DrawPicture ( this_class->step_pict_handle, &bounds );
    else
        EraseRect ( &bounds );

    SetClip( saved_clip_rgn );
    DisposeRgn( saved_clip_rgn );

    // SetColorAndPenState ( &state );
    // SetThemeDrawingState( state, true );
    // SetGWorld ( port, gdh );
}
```

```
//  
// _____  
// _____  
// _____  
  
#ifndef __o_base_dialog__  
#define __o_base_dialog__  
  
#include "o_base_window.h"  
#include "my_dialogs.h"  
  
// _____  
  
//class o_base_dialog //: public o_base_window  
class o_base_dialog : public o_base_window  
{  
    // constructors & destructors  
private:  
    o_base_dialog ();  
  
public:  
    o_base_dialog ( short );  
    virtual ~o_base_dialog ();  
  
    // drawing and clicking  
    static pascal OSStatus do_Process_Content_Click ( EventHandlerCallRef, EventRef, void* );  
    virtual Boolean do_Handle_Content_Click ( EventRecord * );  
    virtual void do_Handle_Item_Hit ( short );  
  
    // window state manipulation  
    virtual Boolean do_Update_Cursor ( EventRecord *, RgnHandle );  
  
    // other info  
    virtual Boolean do_Is_Visible ();  
  
    // positioning and frame info  
    virtual Rect do_Get_Port_Rect ();  
    virtual Rect do_Get_Content_Rect ();  
    virtual Rect do_Get_Structure_Rect ();  
    virtual RgnHandle do_Get_Visible_Region (RgnHandle);  
    virtual RgnHandle do_Get_Content_Region (RgnHandle);  
    virtual RgnHandle do_Get_Structure_Region (RgnHandle);  
  
protected:  
    DialogRef dialog_ref;  
    virtual void do_Calc_Best_Size ( Rect * );  
    short orig_items;  
    EventHandlerUPP content_click_upp;  
  
private:  
};  
  
#ifdef __cplusplus  
    extern "C" {  
#endif  
  
Boolean do_Get_Class_From_Dialog( WindowRef, o_base_dialog ** );  
  
#ifdef __cplusplus  
    }  
#endif  
  
#endif /* __o_base_dialog__ */
```

```
// _____ ©1998-2001 bergdesign inc.
// _____

#include "o_base_dialog.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

// _____

o_base_dialog::o_base_dialog ()
{
    DEBUG_PRINT("Entered o_base_dialog::o_base_dialog()");
    dialog_ref = NULL;
    DEBUG_PRINT("Left o_base_dialog::o_base_dialog()");
}

// _____

o_base_dialog::o_base_dialog ( short dlog_id ) .
{
    DEBUG_PRINT("Entered o_base_dialog::o_base_dialog(...)");
    dialog_ref = GetNewDialog ( dlog_id, NULL, (WindowRef)-1L );
    if ( NULL != dialog_ref )
    {
        // save the WindowRef for this dialog in the base class
        window_ref = GetDialogWindow( dialog_ref );
/*
        EventTargetRef target_ref = GetWindowEventTarget(window_ref);
//
        InstallStandardEventHandler(target_ref);

        // install click handler
        EventTypeSpec event_types[1];
        event_types[0].eventClass = kEventClassWindow;
//
        event_types[0].eventKind = kEventWindowHandleContentClick;
        event_types[0].eventKind = kEventWindowClickContentRgn;
*/
        content_click_upp = NULL;
//
        content_click_upp = NewEventHandlerUPP( o_base_dialog::do_Process_Content_Click );
//
        InstallWindowEventHandler( window_ref, content_click_upp, 1, event_types, this, NULL );

        o_base_dialog* this_ptr = this;
        SetWindowProperty( window_ref, '????', 'this', sizeof(o_base_dialog *), &this_ptr );
        SetPortWindowPort( window_ref );
        SelectWindow( window_ref );
    }

    DEBUG_VAR_PRINT("o_base_dialog DialogRef: %#010X",dialog_ref);
    DEBUG_PRINT("Left o_base_dialog::o_base_dialog(...)");
}

// _____

o_base_dialog::~o_base_dialog ()
{
    DEBUG_PRINT("Entered o_base_dialog destructor");

    if ( dialog_ref )
    {
        if( NULL != content_click_upp )
        {
            DisposeEventHandlerUPP( content_click_upp );
            content_click_upp = NULL;
        }

        DisposeDialog ( dialog_ref );
        dialog_ref = NULL;
        window_ref = NULL;
    }

    DEBUG_PRINT("Left o_base_dialog destructor");
}

// _____

#pragma mark -

// _____

pascal OSStatus o_base_dialog::do_Process_Content_Click ( EventHandlerCallRef next_handler, EventRef the_event,
void* user_data )
{
    OSStatus handled = eventNotHandledErr;

```

```
EventRecord event_rec;

    ConvertEventRefToEventRecord( the_event, &event_rec );

    if( ((o_base_dialog *)user_data)->do_Handle_Content_Click( &event_rec ) )
        handled = noErr;
    else
        handled = eventNotHandledErr;

    return( handled );
}

//
Boolean o_base_dialog::do_Handle_Content_Click ( EventRecord *event )
{
    Point          where;
    SInt16         item_hit = NULL;
    ControlHandle   control = NULL;
    ControlPartCode part_code;
    Boolean         click_was_handled = false;

    where = event->where;
    GlobalToLocal ( &where ); // the current port must be correct or GlobalToLocal won't work right

    // When an embedding hierarchy is established, FindDialogItem()
    // returns the deepest control selected by the user corresponding
    // to the point specified in the point parameter. When an embedding
    // hierarchy does not exist, FindDialogItem() performs a linear search
    // of the item list resource and returns a number corresponding to the
    // hit item's position in the item list resource. For example, it returns 0
    // for the first item in the item list, 1 for the second, and 2 for the third.
    // You must add "1" to the value to get the real item number.
    // If the mouse is not over a dialog item, FindDialogItem() returns -1.

    item_hit = FindDialogItem ( dialog_ref, where ) + 1;
    DEBUG_VAR_PRINT("FindDialogItem() returned item_hit = %d",item_hit);

    if ( item_hit )
    {
        GetDialogItemAsControl ( dialog_ref, item_hit, &control );

        if ( control && IsControlActive(control) && IsControlVisible(control) )
        {
            part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
            DEBUG_VAR_PRINT("HandleControlClick() returned part_code = %d",part_code);

            if ( part_code )
            {
                do_Handle_Item_Hit ( item_hit );
                click_was_handled = true;
            }
        }
    }

    // control = FindControlUnderMouse ( where, dialog_window, &part_code );
    //
    // if ( control )
    // {
    //     DEBUG_VAR_PRINT("FindControlUnderMouse() found control = %#010x",control);
    //     DEBUG_EXTRA_VAR_PRINT("", part_code = %d",part_code);
    //
    //     if ( part_code != kControlNoPart && IsControlVisible(control) && IsControlActive(control) )
    //     {
    //         part_code = HandleControlClick ( control, where, event->modifiers, (ControlActionUPP)-1 );
    //         DEBUG_VAR_PRINT("HandleControlClick() returned part_code = %d",part_code);
    //
    //         if ( part_code )
    //         {
    //             short          num_items, i;
    //             ControlHandle   test_control;
    //
    //             num_items = CountDITL ( dialog_window );
    //
    //             for ( i = 1; i <= num_items; i++ )
    //             {
    //                 GetDialogItemAsControl ( dialog_window, i, &test_control );
    //                 if ( test_control == control )
    //                 {
    //                     item_hit = i;
    //                     break;
    //                 }
    //             }
    //
    //             DEBUG_VAR_PRINT("Item number of control = %d",item_hit);
    //
    //             do_Handle_Item_Hit ( item_hit );
    //             click_was_handled = true;
    //         }
    //     }
    // }
}
```

```
//      }  
//    }  
//  }  
  
    return ( click_was_handled );  
}  
  
//  
void o_base_dialog::do_Handle_Item_Hit ( short item_hit )  
{  
    SysBeep(1);  
}  
  
//  
Boolean o_base_dialog::do_Update_Cursor ( EventRecord *the_event, RgnHandle cursor_region )  
{  
    Boolean    changed_cursor = false;  
  
    return ( changed_cursor );  
}  
  
//  
Boolean o_base_dialog::do_Is_Visible ()  
{  
    return( IsWindowVisible( GetDialogWindow( dialog_ref ) ) );  
    // return( IsWindowVisible( window_ref ) );  
}  
  
//  
#pragma mark -  
  
//  
Rect o_base_dialog::do_Get_Port_Rect ()  
{  
    Rect    bounds;  
  
    // Remember - this is in local coordinates  
    GetWindowPortBounds( GetDialogWindow( dialog_ref ), &bounds );  
    // GetWindowPortBounds( window_ref, &bounds );  
  
    return( bounds );  
}  
  
//  
Rect o_base_dialog::do_Get_Content_Rect ()  
{  
    Rect    bounds;  
  
    // Remember - this is in global coordinates  
    GetWindowBounds( GetDialogWindow( dialog_ref ), kWindowContentRgn, &bounds );  
    // GetWindowBounds( window_ref, kWindowContentRgn, &bounds );  
  
    return( bounds );  
}  
  
//  
Rect o_base_dialog::do_Get_Structure_Rect ()  
{  
    Rect    bounds;  
  
    // Remember - this is in global coordinates  
    GetWindowBounds( GetDialogWindow( dialog_ref ), kWindowStructureRgn, &bounds );  
    // GetWindowBounds( window_ref, kWindowStructureRgn, &bounds );  
  
    return(bounds);  
}  
  
//  
RgnHandle o_base_dialog::do_Get_Visible_Region ( RgnHandle the_region )  
{  
    // Remember - this is in local coordinates  
    GetPortVisibleRegion( GetWindowPort( GetDialogWindow( dialog_ref ) ), the_region );  
    // GetPortVisibleRegion( GetWindowPort( window_ref ), the_region );  
  
    return( the_region );  
}  
  
//  
// It's important to note that the GetWindowRegion() function copies  
// the region from the window's structure to your already initialized region.
```



```
// It does not give you a handle to the window's region.

RgnHandle o_base_dialog::do_Get_Content_Region ( RgnHandle the_region )
{
    // Remember - this is in global coordinates
    GetWindowRegion( GetDialogWindow( dialog_ref ), kWindowContentRgn, the_region );
    // GetWindowRegion( window_ref, kWindowContentRgn, the_region );

    return(the_region);
}

// _____

RgnHandle o_base_dialog::do_Get_Structure_Region ( RgnHandle the_region )
{
    // Remember - this is in global coordinates
    GetWindowRegion( GetDialogWindow( dialog_ref ), kWindowStructureRgn, the_region );
    // GetWindowRegion( window_ref, kWindowStructureRgn, the_region );

    return(the_region);
}

// _____

#pragma mark -

// _____

void o_base_dialog::do_Calc_Best_Size ( Rect *best_rect )
{
    GDHandle          gdh;
    unsigned char      gdh_state;

    gdh = GetMainDevice();
    gdh_state = HGetState ( (Handle)gdh );
    HLock ( (Handle)gdh );
    *best_rect = (*gdh)->gdRect;
    HSetState ( (Handle)gdh, gdh_state );
}

// _____

#pragma mark -

// _____
// This function tests a generic window pointer to determine if the window
// is a C++ dialog class of our creation, or a normal toolbox window.
// If it is a C++ window of our design, it returns "true" and the class pointer
// from the refCon field of the given window. If it's a normal toolbox window,
// it returns false.

Boolean do_Get_Class_From_Dialog( WindowRef window_ref, o_base_dialog **dialog_obj )
{
    OSStatus          err = noErr;
    Boolean            is_dialog_object = false;
    o_base_dialog*     this_ptr = NULL;
    UInt32             prop_size = NULL;

    if( window_ref )
    {
        err = GetWindowProperty( window_ref, '????', 'this', sizeof(o_base_dialog*), &prop_size, &this_ptr );
        if( noErr == err )
        {
            if( this_ptr != NULL )
            {
                // The use of virtual functions lets us dereference a base class pointer
                // and have the derived class functions called. Cool.
                *dialog_obj = (o_base_dialog *)this_ptr;
                is_dialog_object = true;
            }
            else
            {
                is_dialog_object = false;
            }
        }
        else
        {
            is_dialog_object = false;
        }
    }

    return( is_dialog_object );
}
```

```

class o_base_window
{
public:

    // constructors & destructors
    o_base_window ();
    o_base_window ( Rect *, Rect *, Rect *, Boolean, WindowAttributes, ThemeBrush, WindowRe
virtual
    ~o_base_window ();

    // drawing and clicking
virtual Boolean    do_Handle_Click ( EventRecord *, short );
virtual Boolean    do_Handle_Content_Click ( EventRecord * );
virtual Boolean    do_Handle_Key_Down ( EventRecord * );
virtual void       do_Key_Down_Post_Processing ();
virtual void       do_Update ();
void              do_Debug_Update_Region ();
virtual void       do_Draw ();
virtual void       do_Force_Update ();
virtual void       do_Force_Draw ();
virtual void       do_Idle ();
void              do_Set_Port ();

    // window state manipulation
virtual OSStatus   do_Show ( Boolean );
virtual void       do_Move ( short, short );
virtual void       do_Move_To ( Point );
virtual Boolean    do_OK_To_Close ();
virtual Boolean    do_Update_Cursor ( EventRecord *, RgnHandle );
virtual void       do_Resize_Content_Rect ( short, short ); // useful for floaters
virtual void       do_Resize_Contents ();

    // other info
virtual void       do_Set_Title ( const unsigned char *title );
virtual void       do_Get_Title ( Str255 title );
virtual Boolean    do_Is_Visible ();

    // positioning and frame info
virtual short      do_Get_Title_Bar_Height ();
virtual Rect       do_Get_Port_Rect ();
virtual Rect       do_Get_Content_Rect ();
virtual Rect       do_Get_Structure_Rect ();
virtual RgnHandle   do_Get_Visible_Region (RgnHandle);
virtual RgnHandle   do_Get_Content_Region (RgnHandle);
virtual RgnHandle   do_Get_Structure_Region (RgnHandle);
virtual RgnHandle   do_Get_Update_Region (RgnHandle);

    WindowPtr       do_Get_Window_Pointer ();
    DialogRef       do_Get_Dialog_From_Window();

protected:

    WindowRef       window_ref;
    ControlHandle    root;
    Rect            min_window_rect;
    Rect            max_window_rect;
    Rect            best_window_rect;

```

```
        void          do_Create_Window ( Rect *, Str255, Boolean, WindowAttributes, ThemeBrush, WindowRef );
        void          do_Dispose_Window ();

    virtual void      do_Calc_Min_Size ( Rect * );
    virtual void      do_Calc_Max_Size ( Rect * );
    virtual Point      do_Calc_Ideal_Size ();

private:

};

// _____

#ifdef __cplusplus
extern "C" {
#endif

Boolean      do_Get_Class_From_Window ( WindowRef, o_base_window ** );

#ifdef __cplusplus
}
#endif

#endif /* __o_base_window__ */
```

```
// _____  
// _____ ©1998-2001 bergdesign inc.  
// _____  
  
#include "o_base_window.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
o_base_window::o_base_window()  
{  
    DEBUG_PRINT("Entered o_base_window::o_base_window(...)");  
  
    window_ref = NULL;  
    // activate_event_handler = NULL;  
    root = NULL;  
  
    min_window_rect.left = 0;  
    min_window_rect.top = 0;  
    min_window_rect.right = kBaseWindowMinWidth;  
    min_window_rect.bottom = kBaseWindowMinHeight;  
  
    max_window_rect.left = 0;  
    max_window_rect.top = 0;  
    max_window_rect.right = kBaseWindowMaxWidth;  
    max_window_rect.bottom = kBaseWindowMaxHeight;  
  
    best_window_rect.left =  
    best_window_rect.top =  
    best_window_rect.right =  
    best_window_rect.bottom = -1;  
  
    DEBUG_PRINT("Left o_base_window::o_base_window(...)");  
}  
  
// _____  
  
o_base_window::o_base_window( Rect *min_bounds, Rect *max_bounds, Rect *best_bounds, Boolean visible,  
                             WindowAttributes attributes, ThemeBrush brush, WindowRef behind )  
{  
    DEBUG_PRINT("Entered o_base_window::o_base_window(...)");  
  
    window_ref = NULL;  
    // activate_event_handler = NULL;  
    root = NULL;  
  
    // If a min rect was specified, we store it. Otherwise, we set the min to  
    // a predetermined constant specified in the header file.  
    if( min_bounds == NULL )  
    {  
        min_window_rect.left = 0;  
        min_window_rect.top = 0;  
        min_window_rect.right = kBaseWindowMinWidth;  
        min_window_rect.bottom = kBaseWindowMinHeight;  
    }  
    else  
    {  
        min_window_rect = *min_bounds;  
    }  
  
    // If a max rect was specified, we store it. Otherwise, we set the max to  
    // the limits imposed by QuickDraw (a signed short of max +32767) which is  
    // also specified in the header file.  
    if( max_bounds == NULL )  
    {  
        max_window_rect.left = 0;  
        max_window_rect.top = 0;  
        max_window_rect.right = kBaseWindowMaxWidth;  
        max_window_rect.bottom = kBaseWindowMaxHeight;  
    }  
    else  
    {  
        max_window_rect = *max_bounds;  
    }  
  
    // If a "best" rect was specified, we store it. Otherwise, we put a flag in our  
    // rect which tells the zoom function to zoom to the max screen size of whatever  
    // screen the window best fits onto.  
    if( best_bounds == NULL )  
    {  
        best_window_rect.left =  
        best_window_rect.top =  
        best_window_rect.right =  
        best_window_rect.bottom = -1;  
    }  
}
```

```
    else
    {
        best_window_rect = *best_bounds;
    }

    do_Create_Window( &min_window_rect, NULL, visible, attributes, brush, behind );

    DEBUG_VAR_PRINT("o_base_window WindowRef: %#010X",window_ref);
    DEBUG_PRINT("Left o_base_window::o_base_window(...)");
}

//
o_base_window::~o_base_window()
{
    DEBUG_PRINT("Entered o_base_window::~o_base_window()");

    do_Dispose_Window();

    DEBUG_PRINT("Left o_base_window::~o_base_window()");
}

//
void o_base_window::do_Create_Window( Rect *bounds, Str255 title, Boolean visible,
                                     WindowAttributes attributes, ThemeBrush brush, WindowRef behind )
{
    OSStatus      err = noErr;
    CGrafPtr      port;
    GDHandle      gdh;
    //RGBColor     black;

    DEBUG_PRINT("Entered o_base_window::do_Create_Window()");

    if( window_ref != NULL )
    {
        DEBUG_PRINT("window_ref not NULL");
        goto bail;
    }

    GetGWorld( &port, &gdh );

    err = CreateNewWindow( kDocumentWindowClass, attributes, bounds, &window_ref );
    if( noErr == err && window_ref != NULL )
    {
        DEBUG_VAR_PRINT("New WindowRef: %#010X",window_ref);

        SetPortWindowPort( window_ref );

        o_base_window* this_ptr = this;
        err = SetWindowProperty( window_ref, '????', 'this', sizeof(o_base_window*), &this_ptr );
        err = SetWindowProperty( window_ref, '????', 'actv', sizeof(ActivateHandlerUPP), &activate_event_handler );

        // We set the window background to a theme color so it doesn't
        // look so plain. Most everything we'll do from here on out will be
        // in an image well of some sort to give the interface a nice 3D look.
        // If the user passes in a bad brush color, we'll pick an acceptable one for them.
        err = SetThemeWindowBackground( (WindowPtr)window_ref, brush, true );
        if( err == appearanceBadBrushIndexErr )
        {
            SetThemeWindowBackground( (WindowPtr)window_ref, kThemeActiveModelessDialogBackgroundBrush, true );
        }

        // black.red = black.green = black.blue = 0x0000;
        // SetWindowContentColor( (WindowPtr)window_ref, &black );

        // We set the window font to a nice font so we can just call
        // DrawString() later without having to worry about it.
        TextFont( applFont );
        TextFace( bold );
        TextSize( 9 );

        // We install the standard window handler in case the attributes
        // didn't include 'kWindowStandardHandlerAttribute'
        if( !(attributes & kWindowStandardHandlerAttribute) )
        {
            EventTargetRef target_ref = GetWindowEventTarget(window_ref);
            InstallStandardEventHandler(target_ref);
        }

        // So we don't have to worry about it later, we create a root control
        // so that Appearance Manager embedding works correctly.
        err = CreateRootControl( (WindowPtr)window_ref, &root );
        if( err )
        {
            DEBUG_PRINT("Could not create root control");
        }
    }
}
```

```
        if( visible == true )
        {
            // If the window is supposed to be visible, we show it,
            // zoom it to full size and make it active.
            do_Show( true );

            if( attributes & kWindowFullZoomAttribute || attributes & kWindowResizableAttribute )
            {
                Point ideal_size = do_Calc_Ideal_Size();
                ZoomWindowIdeal(window_ref, inZoomOut, &ideal_size);
            }

            SelectWindow( window_ref );
        }
        else
        {
            // If the window's not visible, we set the port back
            // to what it was and just sit quietly in the background.
            SetGWorld( port, gdh );
        }
    }
    else
    {
        DEBUG_VAR_PRINT("NewWindowReference() returned error %d",err);
    }
}

bail:
    DEBUG_PRINT("Left o_base_window::do_Create_Window()");
}

//
void o_base_window::do_Dispose_Window()
{
    DEBUG_PRINT("Entered o_base_window::do_Dispose_Window()");

    if( root )
    {
        DisposeControl( root );
        root = NULL;
    }

    if( window_ref )
    {
        DisposeWindow(window_ref);
        window_ref = NULL;
    }

    DEBUG_PRINT("Left o_base_window::do_Dispose_Window()");
}

//
#pragma mark -
//
Boolean o_base_window::do_Handle_Click( EventRecord *event, short part )
{
    Boolean        click_was_handled = false;
    WindowRef      front_window = NULL;
    WindowRef      front_doc_window = NULL;

    DEBUG_PRINT("Entered o_base_window::do_Handle_Click()");

    front_window = FrontWindow();
    front_doc_window = FrontNonFloatingWindow();

    // If a modal dialog is in front of us and we're inactive, we don't do anything.
    if( WindowIsModal( front_window ) && ( window_ref != front_window ) )
    {
        SysBeep(1);
        click_was_handled = true;
    }
    else
    {
        // CGrafPtr    port;
        // GDHandle    gdh;
        RgnHandle    vis_rgn = NewRgn();

        // Since clicks in the content area, window resizing and window zooming all require
        // some use of local coordinates, we need to make sure the port is set properly
        // so that things work correctly.
        // Setting the port here makes it easier for us elsewhere since we won't
        // have to worry about setting the current port in most functions called from here.
        // We'll be able to overload those without having to worry about things.
```

```
//      GetGWorld( &port, &gdh );  
//      SetPortWindowPort( window_ref );  
  
//      DEBUG_VAR_PRINT("GetGWorld( port:%#010X",port);  
//      DEBUG_EXTRA_VAR_PRINT("  gdh:%#010X ",gdh);  
//      DEBUG_VAR_PRINT("SetPortWindowPort( %#010X )",window_ref);  
  
switch( part )  
{  
    case inDesk:          // 0 We shouldn't ever get these.  
    case inMenuBar:       // 1 They're just here for completeness.  
    case inSysWindow:     // 2 And yes, we need to fall through.  
    case inGoAway:        // 6  
    {  
        click_was_handled = false;  
        break;  
    }  
    case inContent: // 3  
    {  
        // We could do a SetClip() here so that we can speed up a forced  
        // drawing scenario. This way we minimize the jumping around and  
        // toolbox calls for setup and teardown.  
  
        // If the click is in the frontmost doc window or the frontmost window  
        // (which will usually be a floater if the app uses them), we immediately  
        // handle the click so we don't slow the user down any more than we have to.  
        if( window_ref == front_doc_window || window_ref == front_window )  
        {  
            SetPortWindowPort( window_ref );  
            click_was_handled = do_Handle_Content_Click( event );  
        }  
        // If it wasn't the frontmost doc window or the frontmost window,  
        // we need to do some window shuffling.  
        else  
        {  
            SelectWindow( window_ref );  
            SetPortWindowPort( window_ref );  
  
            // If we're a floater and we're not in front, we bring the  
            // window to the front and then handle the click as though the window  
            // was already in front. Since all floaters should be considered to be  
            // on the same level, we want the user to get instant response when, say,  
            // clicking on a slider in a floating window. We don't want them to have  
            // to click twice to get the job done.  
            if( WindowIsFloater( window_ref ) )  
            {  
                click_was_handled = do_Handle_Content_Click( event );  
            }  
            // If we're a doc window and we're not in front, we don't want  
            // to do anything other than bring the window to the front. We  
            // don't want to act on the click as though the user wanted to do  
            // something because that kind of behavior is really annoying.  
            else  
            {  
                click_was_handled = true;  
            }  
        }  
    }  
    break;  
}  
case inDrag: // 4  
{  
    // Are we a floater? If so, we can drag live.  
    if( WindowIsFloater( window_ref ) )  
    {  
        if( window_ref != front_window )  
        {  
            SelectWindow( window_ref );  
            SetPortWindowPort( window_ref );  
            do_Update();  
        }  
        else  
        {  
            SetPortWindowPort( window_ref );  
        }  
  
        Point where = event->where;  
        Point old_where = where;  
  
        Rect port_rect = do_Get_Port_Rect();  
  
        Point port_corner;  
        port_corner.h = port_rect.left;  
        port_corner.v = port_rect.top;  
        LocalToGlobal( &port_corner );  
  
        Point drag_port_offset;  
        drag_port_offset.h = where.h - port_corner.h;
```

```
        drag_port_offset.v = where.v - port_corner.v;

#if TARGET_API_MAC_CARBON

    MouseTrackingResult tracking_result = kMouseTrackingMousePressed;
    while ( tracking_result != kMouseTrackingMouseReleased )
    {
        if( where.h != old_where.h || where.v != old_where.v )
        {
            MoveWindow( window_ref, where.h, where.v, false );
        }
        old_where = where;
        GetMouse( &where );
        LocalToGlobal( &where );
        where.h = where.h - drag_port_offset.h;
        where.v = where.v - drag_port_offset.v;

        TrackMouseLocation (NULL, &where, &tracking_result);
    }

#else

    while( StillDown() )
    {
        if( where.h != old_where.h || where.v != old_where.v )
        {
            MoveWindow( window_ref, where.h, where.v, false );
        }
        old_where = where;
        GetMouse( &where );
        LocalToGlobal( &where );
        where.h = where.h - drag_port_offset.h;
        where.v = where.v - drag_port_offset.v;
        do_Update();
        SystemTask();
    }

#endif

    // do_Update_Cursor();
    // click_was_handled = true;
    }
    else
    {
        Rect limit_rect;

        GetRegionBounds( GetGrayRgn(), &limit_rect );
        InsetRect( &limit_rect, kNudgeSlop, kNudgeSlop );
        DragWindow( window_ref, event->where, &limit_rect );
        do_Update_Cursor();

        click_was_handled = true;
    }

    break;
}
case inGrow: // 5
{
    // Although the Rect parameter is in the form of the Rect data type,
    // the four numbers in the structure represent lengths, not screen
    // coordinates. The top, left, bottom, and right fields of the sizeRect
    // parameter specify the minimum vertical measurement (top), the minimum
    // horizontal measurement (left), the maximum vertical measurement (bottom),
    // and the maximum horizontal measurement (right). The minimum measurements
    // must be large enough to allow a manageable rectangle. Because the user
    // cannot ordinarily move the cursor off the screen, you can safely set
    // the upper bounds to the largest possible length (65,535 pixels) when
    // you're using GrowWindow() to follow cursor movements.

    Rect min_rect, max_rect;
    do_Calc_Min_Size( &min_rect );
    do_Calc_Max_Size( &max_rect );

    Rect grow_rect;
    grow_rect.top = min_rect.bottom - min_rect.top;
    grow_rect.bottom = max_rect.bottom - max_rect.top;

    grow_rect.left = min_rect.right - min_rect.left;
    grow_rect.right = max_rect.right - max_rect.left;

    Rect after_rect;
    Boolean was_resized = ResizeWindow( window_ref, event->where, &grow_rect, &after_rect );

    if( was_resized )
    {
        SetPortWindowPort( window_ref );

        // SizeWindow( window_ref, LoWord(new_size), HiWord(new_size), false );
    }
}
```



```
// InvalRgn() will add the region to the window's update region,  
// causing an update event to be posted in the event queue.  
InvalWindowRgn( window_ref, do_Get_Visible_Region(vis_rgn) );  
do_Resize_Contents();  
}  
  
click_was_handled = true;  
  
break;  
}  
case inZoomIn: // 7  
case inZoomOut: // 8  
{  
    Point ideal_size = do_Calc_Ideal_Size();  
  
    if( IsWindowInStandardState(window_ref, &ideal_size, NULL) )  
    {  
        // If IsWindowInStandardState returns true, the window is  
        // currently zoomed out to the standard state, so the mouse-down  
        // event in the zoom box should be interpreted as inZoomIn  
        part = inZoomIn;  
    }  
    else  
    {  
        // If IsWindowInStandardState returns false, the window is  
        // currently zoomed in to the user state, so the mouse-down event  
        // in the zoom box should be interpreted as inZoomOut  
        part = inZoomOut;  
    }  
  
    if( TrackBox( window_ref, event->where, part ) )  
    {  
        ZoomWindowIdeal(window_ref, part, &ideal_size);  
  
        SetPortWindowPort( window_ref );  
  
        // InvalRgn() will add the region to the window's update region,  
        // causing an update event to be posted in the event queue.  
        InvalWindowRgn( window_ref, do_Get_Visible_Region(vis_rgn) );  
        do_Resize_Contents();  
    }  
  
    click_was_handled = true;  
  
    break;  
}  
case inCollapseBox: // 11  
{  
    click_was_handled = true;  
    break;  
}  
}  
  
// DEBUG_VAR_PRINT("SetGWorld( port:%#010X",port);  
// DEBUG_EXTRA_VAR_PRINT(", gdh:%#010X ",gdh);  
// SetGWorld( port, gdh );  
  
if(vis_rgn)  
    DisposeRgn(vis_rgn);  
}  
  
DEBUG_PRINT("Left o_base_window::do_Handle_Click()");  
  
return(click_was_handled);  
}  
  
// _____  
Boolean o_base_window::do_Handle_Content_Click( EventRecord *event )  
{  
    Boolean click_was_handled = false;  
  
    return( click_was_handled );  
}  
  
// _____  
#pragma mark -  
  
// _____  
Boolean o_base_window::do_Handle_Key_Down( EventRecord *event )  
{  
    Boolean key_was_handled = false;  
  
    return( key_was_handled );  
}
```

```
//  
  
void o_base_window::do_Key_Down_Post_Processing()  
{  
    // If we need to respond to key down events in fields  
    // after they have happened, we can call this function from  
    // do_Handle_Key_Down() after we're done with the keyDown  
    // event. This lets us do things like disable  
    // controls if certain criteria are not met.  
}  
  
//  
  
#pragma mark -  
  
//  
// Update events are automatically generated by the Window Manager  
// when the contents of a window have been pooped on or revealed  
// by moving other windows previously on top of the window. This  
// pooping and revealing adds regions to a window's update region.  
// When the update region is not empty, the Window Manager puts an update  
// event into the event queue which specifies which window needs to be  
// redrawn. The event loop will continue to get update events until  
// the window's drawing routine is called, bracketed by calls to  
// BeginUpdate() and EndUpdate(). The BeginUpdate()/EndUpdate() calls  
// are responsible for clearing a window's update region.  
  
void o_base_window::do_Update()  
{  
    // If we're using only controls, we don't need the GetPort()/SetPort()  
    // and GetColorAndPenState() stuff. But if we do any QuickDraw drawing, we do.  
  
    DEBUG_VAR_PRINT("Entered o_base_window::do_Update(%#010X)", window_ref);  
  
    // Should use GetPort() under OS X?  
    CGrafPtr port;  
    GDHandle gdh;  
    GetGWorld( &port, &gdh );  
    SetPortWindowPort( window_ref );  
  
    ThemeDrawingState state;  
    GetThemeDrawingState( &state );  
  
    // The BeginUpdate() routine automatically sets the clipping region  
    // to the update region which is the visible portion of the invalid  
    // region of the window. If we call do_Update() to force the redraw  
    // of a window that has an empty update region, nothing will be drawn.  
    // We have to first invalidate the region that we want to have drawn.  
  
    BeginUpdate( (WindowPtr)window_ref );  
  
    // do_Debug_Update_Region();  
    do_Draw();  
    // QDFlushPortBuffer( GetWindowPort(window_ref), the_rgn );  
  
    EndUpdate( (WindowPtr)window_ref );  
  
    SetThemeDrawingState( state, true );  
    SetGWorld( port, gdh );  
  
    DEBUG_PRINT("Left o_base_window::do_Update()");  
}  
  
//  
  
void o_base_window::do_Debug_Update_Region()  
{  
    #if !OPAQUE_TOOLBOX_STRUCTS  
        DEBUG_VAR_PRINT("The update region is bounded by  
%d", (**((WindowPeek)window_ref)->updateRgn)).rgnBBox.left);  
        DEBUG_EXTRA_VAR_PRINT(" %d", (**((WindowPeek)window_ref)->updateRgn)).rgnBBox.top);  
        DEBUG_EXTRA_VAR_PRINT(" %d", (**((WindowPeek)window_ref)->updateRgn)).rgnBBox.right);  
        DEBUG_EXTRA_VAR_PRINT(" %d", (**((WindowPeek)window_ref)->updateRgn)).rgnBBox.bottom);  
    #else  
        RgnHandle the_rgn = NewRgn();  
        if( NULL != the_rgn )  
        {  
            Rect    the_rect;  
  
            GetRegionBounds( do_Get_Update_Region( the_rgn ), &the_rect );  
            DEBUG_VAR_PRINT("The update region is bounded by l,t,r,b %d", the_rect.left);  
            DEBUG_EXTRA_VAR_PRINT(" %d", the_rect.top);  
            DEBUG_EXTRA_VAR_PRINT(" %d", the_rect.right);  
            DEBUG_EXTRA_VAR_PRINT(" %d", the_rect.bottom);  
            DisposeRgn( the_rgn );  
        }  
    #endif  
}
```

```
#endif

// SysBeep(1);
}

//
// This is our drawing function for the window's content. It is the
// function that we will override the most. We separated it from
// the do_Update() routine so that we don't need to worry about SetPort()
// stuff everytime we override it. We can just assume everything is ok
// by this point and start drawing.

void o_base_window::do_Draw()
{
    DEBUG_PRINT("Entered o_base_window::do_Draw()");

    // We could leave this function empty and overload it in all of
    // our derived classes, but if we use custom Appearance Manager
    // controls for most of our dirty work, UpdateControls() is
    // pretty much all we need to draw in our derived class windows.
    // We only need to override this routine if we need CopyBits()
    // or other non-control manager drawing functions.
    // So, we just put UpdateControls() in here and don't
    // worry about drawing functions most of the time.

    RgnHandle vis_rgn = NewRgn();
    if( NULL != vis_rgn )
    {
        DEBUG_PRINT("Updating controls...");
        UpdateControls( (WindowPtr)window_ref, do_Get_Visible_Region(vis_rgn) );
        UpdateControls( (WindowPtr)window_ref, do_Get_Update_Region(vis_rgn) );
        DisposeRgn(vis_rgn);
    }

    DEBUG_PRINT("Left o_base_window::do_Draw()");
}

//
void o_base_window::do_Force_Update()
{
    CGrafPtr      port;
    GDHandle      gdh;

    DEBUG_PRINT("Entered o_base_window::do_Force_Update()");

    GetGWorld( &port, &gdh );
    SetPortWindowPort( window_ref );

    // InvalRgn() will add the region to the window's update region,
    // causing an update event to be posted in the event queue.
    RgnHandle vis_rgn = NewRgn();
    if( NULL != vis_rgn )
    {
        Rect      the_rect;

        InvalWindowRgn( window_ref, do_Get_Visible_Region(vis_rgn) );

        // QDFlushPortBuffer( GetWindowPort(window_ref), vis_rgn );
        /*
        RgnHandle dirty_rgn = NewRgn();
        if( NULL != dirty_rgn )
        {
            QDGetDirtyRegion( GetWindowPort(window_ref), dirty_rgn );
            UnionRgn( vis_rgn, dirty_rgn, dirty_rgn );
            QDSetDirtyRegion( GetWindowPort(window_ref), dirty_rgn );
            DisposeRgn( dirty_rgn );
        }
        */
        GetRegionBounds( vis_rgn, &the_rect );
        DEBUG_VAR_PRINT("The visible region is bounded by %d",the_rect.left);
        DEBUG_EXTRA_VAR_PRINT(", %d",the_rect.top);
        DEBUG_EXTRA_VAR_PRINT(", %d",the_rect.right);
        DEBUG_EXTRA_VAR_PRINT(", %d",the_rect.bottom);

        GetRegionBounds( do_Get_Update_Region( vis_rgn ), &the_rect );
        DEBUG_VAR_PRINT("The update region is now bounded by %d",the_rect.left);
        DEBUG_EXTRA_VAR_PRINT(", %d",the_rect.top);
        DEBUG_EXTRA_VAR_PRINT(", %d",the_rect.right);
        DEBUG_EXTRA_VAR_PRINT(", %d",the_rect.bottom);

        DisposeRgn(vis_rgn);
    }

    SetGWorld( port, gdh );

    DEBUG_PRINT("Left o_base_window::do_Force_Update()");
}
```

```
//
// We need a do_Force_Draw() function for operations that need speed.
// We could call the InvalRgn() function then the do_Update() function,
// but these cause update events to be generated and require a lot of jumping
// in and out of toolbox calls which will waste precious time. This
// is not the most efficient routine, but it is a good balance between speed
// and simplicity and cleanliness. Here we just set the clipRgn to
// the visRgn and call do_Draw() so we waste as little time as possible.

void o_base_window::do_Force_Draw()
{
    CGrafPtr      port;
    GDHandle      gdh;
    RgnHandle      saved_clip_rgn = NewRgn();
    RgnHandle      vis_clip_rgn = NewRgn();

    DEBUG_PRINT("Entered o_base_window::do_Force_Draw()");

    GetGWorld( &port, &gdh );
    SetPortWindowPort( window_ref );

    // Remember that GetClip() copies the region out of the window record
    // and into the region specified, so you need to create a new region to
    // hold the data, not just a RgnHandle. But with SetClip(), we're simply
    // pointing to the existing visRgn of the window, so it just gets copied
    // into the clipRgn with no additional memory allocation needed.

    GetClip( saved_clip_rgn );

    do_Get_Visible_Region( vis_clip_rgn );

    SetClip( vis_clip_rgn );

    do_Draw();

    SetClip( saved_clip_rgn );

    if( saved_clip_rgn )
        DisposeRgn( saved_clip_rgn );

    if( vis_clip_rgn )
        DisposeRgn( vis_clip_rgn );

    SetGWorld( port, gdh );

    DEBUG_PRINT("Left o_base_window::do_Force_Draw()");
}

//
void o_base_window::do_Idle()
{
    if( window_ref )
        IdleControls( window_ref );
}

//
Boolean o_base_window::do_Update_Cursor( EventRecord *the_event, RgnHandle cursor_region )
{
    Boolean      changed_cursor = false;

    return( changed_cursor );
}

//
void o_base_window::do_Set_Port()
{
    if( window_ref )
        SetPortWindowPort( window_ref );
}

//
#pragma mark -

//
OSStatus o_base_window::do_Show( Boolean show )
{
    OSStatus err = noErr;

    if( show )
    {
        err = TransitionWindow( window_ref, kWindowZoomTransitionEffect, kWindowShowTransitionAction, NULL );
    }
}
```

```
    else
    {
        err = TransitionWindow( window_ref, kWindowZoomTransitionEffect, kWindowHideTransitionAction, NULL );
    }
    return( err );
}

//
void o_base_window::do_Move( short h_change, short v_change )
{
    if( window_ref )
    {
        Rect is = do_Get_Port_Rect();
        short new_left = is.left + h_change;
        short new_top = is.top + v_change;

        // We need to make sure that the last, bring-to-front,
        // parameter of MoveWindow() doesn't ever get set to true.
        // This would cause a call to SelectWindow() which would
        // generate an activate event.

        MoveWindow( (WindowPtr)window_ref, new_left, new_top, false );
    }
}

//
void o_base_window::do_Move_To( Point new_top_left )
{
    if( window_ref )
        MoveWindow( (WindowPtr)window_ref, new_top_left.h, new_top_left.v, false );
}

//
#pragma mark -

//
// These functions allow you to override the default window
// size restrictions in a derived class. This might be useful
// if you need to dynamically change the restrictions.

void o_base_window::do_Calc_Min_Size( Rect *min_rect )
{
    *min_rect = min_window_rect;
}

//
void o_base_window::do_Calc_Max_Size( Rect *max_rect )
{
    *max_rect = max_window_rect;
}

//
Point o_base_window::do_Calc_Ideal_Size()
{
    Point ideal_size; // point where h and v are ideal width and height of the content region
    Rect best_rect;

    // If a "best" rect was not specified for the window, we assume that the
    // max screen size is the best size. Otherwise, we use the specified rectangle.
    if( best_window_rect.left == -1 &&
        best_window_rect.top == -1 &&
        best_window_rect.right == -1 &&
        best_window_rect.bottom == -1 )
    {
        GDHandle gdh;
        unsigned char gdh_state;

        gdh = GetMainDevice();
        gdh_state = HGetState( (Handle)gdh );
        HLock( (Handle)gdh );

        best_rect = (*gdh)->gdRect;

        HSetState( (Handle)gdh, gdh_state );
    }
    else
    {
        best_rect = best_window_rect;
    }

    ideal_size.h = best_rect.right - best_rect.left;
    ideal_size.v = best_rect.bottom - best_rect.top;
}
```

```
        return(ideal_size);
    }

//
void o_base_window::do_Resize_Content_Rect( short h, short v )
{
    if( h > 0 && v > 0 )
    {
        CGrafPtr    port;
        GDHandle     gdh;
        RgnHandle     vis_rgn = NewRgn();

        GetGWorld( &port, &gdh );
        SetPortWindowPort( window_ref );

        SizeWindow( window_ref, h, v, false );
        // InvalRgn() will add the region to the window's update region,
        // causing an update event to be posted in the event queue.
        InvalWindowRgn( window_ref, do_Get_Visible_Region(vis_rgn) );
        do_Resize_Contents();

        SetGWorld( port, gdh );

        if(vis_rgn)
            DisposeRgn(vis_rgn);
    }
}

//
void o_base_window::do_Resize_Contents()
{
}

//
#pragma mark -

//
Boolean o_base_window::do_OK_To_Close()
{
    Boolean    ok_to_close = true;

    return( ok_to_close );
}

//
void o_base_window::do_Get_Title( Str255 title )
{
    GetWTitle( (WindowPtr)window_ref, title );
}

//
void o_base_window::do_Set_Title( const unsigned char *title )
{
    SetWTitle( (WindowPtr)window_ref, title );
}

//
Boolean o_base_window::do_Is_Visible()
{
    return( IsWindowVisible(window_ref) );
}

//
#pragma mark -

//
short o_base_window::do_Get_Title_Bar_Height()
{
    short    height;

    height = ( do_Get_Structure_Rect() ).top - ( do_Get_Content_Rect() ).top;
    return( height );
}

//
Rect o_base_window::do_Get_Port_Rect()
{
}
```

```
// Remember - this is in local coordinates
#if !OPAQUE_TOOLBOX_STRUCTS
    return ( ((WindowPeek)window_ref)->port.portRect );
#else
    Rect    bounds;
    GetWindowPortBounds( window_ref, &bounds );
    return( bounds );
#endif
}

//
Rect o_base_window::do_Get_Content_Rect()
{
    // Remember - this is in global coordinates
    #if !OPAQUE_TOOLBOX_STRUCTS
        return ( (*( do_Get_Content_Region() ) ).rgnBBox );
    #else
        Rect    bounds;
        GetWindowBounds( window_ref, kWindowContentRgn, &bounds );
        return( bounds );
    #endif
}

//
Rect o_base_window::do_Get_Structure_Rect()
{
    // Remember - this is in global coordinates
    #if !OPAQUE_TOOLBOX_STRUCTS
        return ( (*( do_Get_Structure_Region() ) ).rgnBBox );
    #else
        Rect    bounds;
        GetWindowBounds( window_ref, kWindowStructureRgn, &bounds );
        return(bounds);
    #endif
}

//
RgnHandle o_base_window::do_Get_Visible_Region()
{
    // Remember - this is in local coordinates
    return ( ((WindowPeek)window_ref)->port.visRgn );
}
#else
RgnHandle o_base_window::do_Get_Visible_Region( RgnHandle the_region )
{
    // Remember - this is in local coordinates
    return( GetPortVisibleRegion( GetWindowPort(window_ref), the_region ) );
}
#endif

//
// It's important to note that the GetWindowRegion() function copies
// the region from the window's structure to your already initialized region.
// It does not give you a handle to the window's region.
#if !OPAQUE_TOOLBOX_STRUCTS
RgnHandle o_base_window::do_Get_Content_Region()
{
    // Remember - this is in global coordinates
    return ( ((WindowPeek)window_ref)->contRgn );
}
#else
RgnHandle o_base_window::do_Get_Content_Region( RgnHandle the_region )
{
    // Remember - this is in global coordinates
    GetWindowRegion( window_ref, kWindowContentRgn, the_region );
    return( the_region );
}
#endif

//
RgnHandle o_base_window::do_Get_Structure_Region()
{
    // Remember - this is in global coordinates
    return ( ((WindowPeek)window_ref)->strucRgn );
}
#else
RgnHandle o_base_window::do_Get_Structure_Region( RgnHandle the_region )
{
    // Remember - this is in global coordinates
    GetWindowRegion( window_ref, kWindowStructureRgn, the_region );
    return( the_region );
}
```

```
}
#endif

//
RgnHandle o_base_window::do_Get_Update_Region( RgnHandle the_region )
{
    // Remember - this is in global coordinates
    GetWindowRegion( window_ref, kWindowUpdateRgn, the_region );
    return( the_region );
}

//
WindowPtr o_base_window::do_Get_Window_Pointer()
{
    // Be careful with this...
    return( (WindowPtr>window_ref );
}

//
DialogRef o_base_window::do_Get_Dialog_From_Window()
{
    return( GetDialogFromWindow( window_ref ) );
}

//
#pragma mark -

//
// This function tests a generic window pointer to determine if the window
// is a C++ window class of our creation, or a normal toolbox window.
// If it is a C++ window of our design, it returns "true" and the class pointer
// from the refCon field of the given window. If it's a normal toolbox window,
// it returns false.
Boolean do_Get_Class_From_Window( WindowRef window_ref, o_base_window **window_obj )
{
    OSStatus      err = noErr;
    Boolean        is_window_object = false;
    o_base_window* this_ptr = NULL;
    UInt32         prop_size = NULL;

    if( window_ref )
    {
        err = GetWindowProperty( window_ref, '????', 'this', sizeof(o_base_window*), &prop_size, &this_ptr );
        if( noErr == err )
        {
            if( this_ptr != NULL )
            {
                // The use of virtual functions lets us dereference a base class pointer
                // and have the derived class functions called. Cool.
                *window_obj = (o_base_window *)this_ptr;
                is_window_object = true;
            }
            else
            {
                is_window_object = false;
            }
        }
        else
        {
            is_window_object = false;
        }
    }

    return( is_window_object );
}
```



```
// _____  
// _____ ©1998-2002 bergdesign inc.  
// _____  
  
#ifndef __my_alerts__  
#define __my_alerts__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Dialogs.h>  
#include <MacTypes.h>  
#include <Sound.h>  
#include <NumberFormatting.h>  
#endif  
#endif  
  
#include "my_windows.h"  
#include "my_dialogs.h"  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
#define kAlertStringFatalErrorError "\pSorry. A fatal error occurred."  
#define kAlertStringFatalErrorExplanation "\pWe have no idea why."  
  
#define kAlertStringLogFileOpenErrorError "\pCannot open the log file."  
#define kAlertStringLogFileOpenErrorExplanation "\pThe log file is in use by another application."  
  
#define kAlertStringNoWindowPointerError "\pA new window could not be created."  
#define kAlertStringNoWindowPointerExplanation "\pA null window pointer was returned."  
  
#define kAlertStringNoMemoryHandleError "\pCouldn't get a new memory handle."  
#define kAlertStringNoMemoryHandleExplanation "\pInsufficient memory available."  
  
#define kAlertStringNewGWorldError "\pCouldn't create a new GWorld."  
#define kAlertStringNewGWorldExplanation "\pInsufficient memory available."  
  
#define kAlertStringGWorldLockError "\pCould not lock the GWorld pixmap."  
#define kAlertStringGWorldLockExplanation "\pDon't know why. Just couldn't."  
  
// standard alert types from Apple headers  
  
// kAlertStopAlert = 0,  
// kAlertNoteAlert = 1,  
// kAlertCautionAlert = 2,  
// kAlertPlainAlert = 3  
  
// returned in itemHit parameter of StandardAlert  
  
// kAlertStdAlertOKButton = 1,  
// kAlertStdAlertCancelButton = 2,  
// kAlertStdAlertOtherButton = 3,  
// kAlertStdAlertHelpButton = 4  
  
void do_Alert_If_Error ( const unsigned char *, OSErr );  
void do_Alert_If_Fatal_Error ( const unsigned char *, OSErr );  
  
SInt16 do_One_Button_Alert ( AlertType, const unsigned char *, const unsigned char *, const unsigned char * );  
SInt16 do_Two_Button_Alert ( AlertType, const unsigned char *, const unsigned char *, const unsigned char *,  
const unsigned char * );  
SInt16 do_Save_Changes_Alert ( const unsigned char * );  
  
#ifdef __cplusplus  
}  
#endif  
  
#endif /* __my_alerts__ */
```



```
    if ( err != noErr )
        return err;
    else
        return item_hit;
}

//
SInt16 do_Two_Button_Alert ( AlertType type,
                             const unsigned char *the_error,
                             const unsigned char *explanation,
                             const unsigned char *default_text,
                             const unsigned char *cancel_text )
{
    AlertStdAlertParamRec    params;
    SInt16                   item_hit;
    CGrafPtr                 port;
    GDHandle                  gdh;
    OSErr                    err = noErr;

    params.movable           = false;
    params.helpButton        = false;
    params.filterProc         = NULL;
    params.defaultText        = default_text;
    params.cancelText         = cancel_text;
    params.otherText          = NULL;
    params.defaultButton      = kAlertStdAlertOKButton;
    params.cancelButton       = kAlertStdAlertCancelButton;
    params.position           = kWindowDefaultPosition;

    GetGWorld ( &port, &gdh );
    // SetCursor ( &qd.arrow );

    DeactivateFloatersAndFirstDocumentWindow();

    err = StandardAlert ( type, the_error, explanation, &params, &item_hit );

    ActivateFloatersAndFirstDocumentWindow();

    // do_Update_Cursor ();
    SetGWorld ( port, gdh );

    if ( err != noErr )
        return err;
    else
        return item_hit;
}

//
SInt16 do_Save_Changes_Alert ( const unsigned char *prompt )
{
    AlertType                 type;
    StringPtr                 explanation;
    AlertStdAlertParamRec     params;
    SInt16                    item_hit;
    CGrafPtr                  port;
    GDHandle                   gdh;
    OSErr                     err = noErr;

    type = kAlertCautionAlert;
    explanation = NULL;
    params.movable           = false;
    params.helpButton        = false;
    params.filterProc         = NULL;
    params.defaultText        = "\pSave";
    params.cancelText         = "\pCancel";
    params.otherText          = "\pDon't Save";
    params.defaultButton      = kAlertStdAlertOKButton;
    params.cancelButton       = kAlertStdAlertCancelButton;
    params.position           = kWindowDefaultPosition;

    GetGWorld ( &port, &gdh );
    // SetCursor ( &qd.arrow );

    DeactivateFloatersAndFirstDocumentWindow();

    err = StandardAlert ( type, prompt, explanation, &params, &item_hit );

    ActivateFloatersAndFirstDocumentWindow();

    // do_Update_Cursor ();
    SetGWorld ( port, gdh );

    if ( err != noErr )
        return err;
    else
```

```
        return item_hit;
    }

//
//void do_Error_Alert ( long error )
//{
//StringPtr      res_string;
//short          i = 1;
//
// while ( res_string[0] != 0 )
// {
//     GetIndString ( res_string, 129, i );
//     StringToNum ();
// }
//}

//
//void do_Create ()
//{
//Handle          str_handle = NULL;
//Ptr             str_pointer = NULL;
//short           current_resource_ID;
//
// if ( !str_handle )
// {
//     // Get the current resource file so we can restore it later.
//     current_resource_ID = CurResFile();
//
//     // Set the resource file to the application.
//     UseResFile( LMGetCurApRefNum() );
//
//     // Get a handle to the STR# resource of the application.
//     str_handle = Get1Resource ( 'STR#', 129 );
//
//     if ( str_handle )
//     {
//         HNoPurge ( str_handle );
//
//         // Count number of strings
//         // Get num bytes strings take up
//         // Allocate memory for array of pointers
//         // Sort pointers
//     }
//
//     // Restore the previous resource file
//     UseResFile ( current_resource_ID );
// }
//
// if ( !ResError() )
// {
//     HLock ( str_handle );
//     str_pointer = *str_handle;
//     if ( str_pointer != NULL )
//     {
//     }
//
//     HUnlock ( str_handle );
//     ReleaseResource ( str_handle );
//     str_handle = NULL;
//     str_pointer = NULL;
// }
//}
```

```
//  
//  
//  
#ifndef __my_apple_events__  
#define __my_apple_events__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <AppleEvents.h>  
#include <Processes.h>  
#endif  
#endif  
  
#include "my_macros.h"  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
// Other includes  
  
OSErr do_Remove_AE_Handler( AEEventClass, AEEventID );  
  
pascal OSErr do_Find_Process_By_Signature( const OSType targetType, const OSType targetCreator,  
ProcessSerialNumberPtr psnPtr );  
static OSErr do_Launch_Process_By_Signature(const OSType pTargetType,const OSType  
pTargetCreator,ProcessSerialNumberPtr psnPtr);  
  
static OSErr do_Search_Volumes(const OSType pTargetType,const OSType pTargetCreator,FSSpec* pFSSpecPtr);  
static OSErr do_Search_Volume(const SInt16 pVRefNum,const OSType pTargetType,const OSType pTargetCreator,FSSpec*  
pFSSpecPtr);  
  
static OSErr do_Find_DTDB_APPL(const OSType pTargetCreator,FSSpec* pFSSpecPtr);  
  
#ifdef __cplusplus  
}  
#endif  
  
#endif /* __my_apple_events__ */
```

```
// _____  
// _____ ©1998-1999 bergdesign inc.  
// _____  
  
#include "my_apple_events.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
OSErr do_Remove_AE_Handler( AEEEventClass the_event_class, AEEEventID the_event_id )  
{  
    AEEEventHandlerUPP ae_handler = NULL;  
    long ae_ref_con = 0L;  
    OSErr err = noErr;  
  
    // We didn't bother keeping the routine descriptor pointers around, so  
    // we just retrieve them from the dispatch table when getting rid of them.  
  
    err = AEGetEventHandler( the_event_class, the_event_id, &ae_handler, &ae_ref_con, false );  
    if( err == noErr && ae_handler != NULL )  
    {  
        err = AERemoveEventHandler( the_event_class, the_event_id, ae_handler, false );  
        DisposeAEEEventHandlerUPP( ae_handler );  
    }  
  
    return( err );  
}  
  
// _____  
  
#pragma mark -  
  
// _____  
  
pascal OSErr do_Find_Process_By_Signature( const OSType targetType, const OSType targetCreator,  
ProcessSerialNumberPtr psnPtr )  
{  
    OSErr anErr = noErr;  
    Boolean lookingForProcess = true;  
  
    ProcessInfoRec infoRec;  
  
    infoRec.processInfoLength = sizeof( ProcessInfoRec );  
    infoRec.processName = nil;  
    infoRec.processAppSpec = nil;  
  
    psnPtr->lowLongOfPSN = kNoProcess;  
    psnPtr->highLongOfPSN = kNoProcess;  
  
    while ( lookingForProcess )  
    {  
        anErr = GetNextProcess( psnPtr );  
        if ( anErr != noErr )  
        {  
            lookingForProcess = false;  
        }  
        else  
        {  
            anErr = GetProcessInformation( psnPtr, &infoRec );  
            if ( ( anErr == noErr )  
                && ( infoRec.processType == targetType )  
                && ( infoRec.processSignature == targetCreator ) )  
            {  
                lookingForProcess = false;  
            }  
        }  
    }  
  
    return(anErr);  
}  
  
// _____  
  
OSErr do_Launch_Process_By_Signature( const OSType pTargetType, const OSType pTargetCreator,  
ProcessSerialNumberPtr psnPtr )  
{  
    FSSpec appFSSpec;  
    OSErr anErr = do_Find_Process_By_Signature( pTargetType, pTargetCreator, psnPtr );  
  
    if (anErr == noErr) // already running  
        return anErr;  
  
    // It's not already running, so look for it (as an APPL) in the desktop database  
    if (pTargetType == 'APPL')  
        anErr = do_Find_DTDB_APPL(pTargetCreator,&appFSSpec);  
}
```

```
// If we haven't found it yet, search all the volumes
if (anErr != noErr)
    anErr = do_Search_Volumes(pTargetType, pTargetCreator, &appFSSpec);

// we found it, so try to launch it
if (anErr == noErr)
{
    LaunchParamBlockRec tLaunchPB;

    tLaunchPB.launchBlockID = extendedBlock;
    tLaunchPB.launchEPBLength = extendedBlockLen;
    tLaunchPB.launchFileFlags = nil;
    tLaunchPB.launchControlFlags = launchContinue + launchNoFileFlags + launchDontSwitch;
    tLaunchPB.launchAppSpec = &appFSSpec;

    anErr = LaunchApplication(&tLaunchPB);
    if (anErr == noErr)
        *psnPtr = tLaunchPB.launchProcessSN;
}

return(anErr);
}

//
#pragma mark -
//

static OSErr do_Search_Volumes( const OSType pTargetType, const OSType pTargetCreator, FSSpec* pFSSpecPtr )
{
    SInt16 index;
    OSErr anErr = noErr;

    for (index = 1; index++; // for each volume...
    {
        XVolumeParam tXVPV;

        tXVPV.ioCompletion = nil;
        tXVPV.ioNamePtr = nil;
        tXVPV.ioVolIndex = index;

        anErr = PBXGetVolInfoSync(&tXVPV); // get its ioVRefNum
        if (anErr == nsvErr) // if no such volume...
            anErr = afpItemNotFound; // ...return application information not found
        if (anErr != noErr) // on error...
            break; // ...break

        anErr = do_Search_Volume(tXVPV.ioVRefNum, pTargetType, pTargetCreator, pFSSpecPtr);
        if (anErr == noErr) // if we found it...
            break; // ...break
    }

    return(anErr);
}

//

static OSErr do_Search_Volume( const SInt16 pVRefNum, const OSType pTargetType, const OSType pTargetCreator,
FSSpec* pFSSpecPtr )
{
    CParamPtr tCParamPtr;
    OSErr anErr;

    if (!pFSSpecPtr)
        return paramErr;

    tCParamPtr = (CParamPtr) NewPtrClear(sizeof(CParam));
    if (tCParamPtr == nil)
        return MemError();

    // initialize the parameter block
    tCParamPtr->ioVRefNum = pVRefNum;
    tCParamPtr->ioMatchPtr = pFSSpecPtr;
    tCParamPtr->ioSearchBits = fsSBFlFndrInfo;

    tCParamPtr->ioReqMatchCount = 1; // only looking for 1
    tCParamPtr->ioSearchTime = 0; // no timeout

    tCParamPtr->ioSearchInfo1 = (CInfoPBPtr) NewPtrClear(sizeof(CInfoPBRec));
    tCParamPtr->ioSearchInfo2 = (CInfoPBPtr) NewPtrClear(sizeof(CInfoPBRec));

    if (tCParamPtr->ioSearchInfo1 && tCParamPtr->ioSearchInfo2)
    {
        // Now see if we can create an 2K optimization buffer
        tCParamPtr->ioOptBuffer = NewPtr(2048);
        if (tCParamPtr->ioOptBuffer)
    }
}
```

```
tCSPParamPtr->ioOptBufSize = 2048;
else
    tCSPParamPtr->ioOptBufSize = 0; // no buffer, sorry

tCSPParamPtr->ioSearchInfo1->hFileInfo.ioNamePtr = nil;
tCSPParamPtr->ioSearchInfo2->hFileInfo.ioNamePtr = nil;
tCSPParamPtr->ioSearchInfo1->hFileInfo.ioFlFndrInfo.fdType = pTargetType;
tCSPParamPtr->ioSearchInfo1->hFileInfo.ioFlFndrInfo.fdCreator = pTargetCreator;

tCSPParamPtr->ioSearchInfo2->hFileInfo.ioFlFndrInfo.fdCreator = 0xFFFFFFFF;
tCSPParamPtr->ioSearchInfo2->hFileInfo.ioFlFndrInfo.fdType = 0xFFFFFFFF;

anErr = PBCatSearchSync(tCSPParamPtr); // search sync
if ((anErr != noErr) || (tCSPParamPtr->ioActMatchCount == 0))
//
}
else
{
    anErr = MemError();
}

// no matter what happened, kill all the memory we allocated
if (tCSPParamPtr->ioSearchInfo1)
    DisposePtr((Ptr)tCSPParamPtr->ioSearchInfo1);

if (tCSPParamPtr->ioSearchInfo2)
    DisposePtr((Ptr)tCSPParamPtr->ioSearchInfo2);

if (tCSPParamPtr->ioOptBuffer)
    DisposePtr((Ptr)tCSPParamPtr->ioOptBuffer);

DisposePtr((Ptr)tCSPParamPtr);

return(anErr);
}

//
static OSErr do_Find_DTDB_APPL( const OSType pTargetCreator, FSSpec* pFSSpecPtr )
{
    SInt16 index;
    OSErr anErr = noErr;

    for (index = 1;;index++) // for each volume...
    {
        XVolumeParam tXVPV;
        DTPBRec deskTopDBRec;

        tXVPV.ioCompletion = nil;
        tXVPV.ioNamePtr = nil;
        tXVPV.ioVolIndex = index;

        anErr = PBXGetVolInfoSync(&tXVPV); // get its ioVRefNum
        if (anErr == nsvErr) // if no such volume...
            anErr = afpItemNotFound; // ...return application information not found
        if (anErr != noErr) // on error...
            break; // ...break

        // now get the DTDB for this volume
        deskTopDBRec.ioNamePtr = nil;
        deskTopDBRec.ioVRefNum = tXVPV.ioVRefNum;
        anErr = PBDTGetPath( &deskTopDBRec );
        if (anErr != noErr)
            break;

        // look in this DTDB for the app with this creator
        deskTopDBRec.ioCompletion = nil;
        deskTopDBRec.ioNamePtr = pFSSpecPtr->name;
        deskTopDBRec.ioIndex = 0;
        deskTopDBRec.ioFileCreator = pTargetCreator;

        anErr = PBDTGetAPPLSync( &deskTopDBRec );
        if (anErr == noErr) // found it
        {
            // stuff our FSSpec
            pFSSpecPtr->vRefNum = deskTopDBRec.ioVRefNum;
            pFSSpecPtr->parID = deskTopDBRec.ioAPPLParID;
            break;
        }
    }

    return(anErr);
}
```



```
//  
//  
//  
#ifndef __my_colorsync__  
#define __my_colorsync__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <MacTypes.h>  
#endif  
#endif  
  
#include "my_macros.h"  
  
#include <stdlib.h>  
#include <stdio.h>  
#include <math.h>  
  
// x + y + z = 1  
struct xyColor  
{  
    float x;  
    float y;  
    float z;  
};  
  
// tristimulus  
struct XYZColor  
{  
    float X;  
    float Y;  
    float Z;  
};  
  
struct xyQuad  
{  
    struct xyColor red;  
    struct xyColor green;  
    struct xyColor blue;  
    struct xyColor white;  
};  
  
struct XYZQuad  
{  
    struct XYZColor red;  
    struct XYZColor green;  
    struct XYZColor blue;  
    struct XYZColor white;  
};  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
    void    do_xy_to_temp( double, double, double * );  
    void    do_temp_to_xy( double *, double *, double );  
  
    int     do_calc_cct( double, double, double * );  
    double  do_max_of_3( double x, double y, double z );  
    void    do_temp_to_XYZ( double, double *, double *, double * );  
  
    void    do_xy_To_XYZ( struct xyQuad *xy, struct XYZQuad *XYZ );  
    void    do_XYZ_To_xy( struct XYZQuad *XYZ, struct xyQuad *xy );  
  
#ifdef __cplusplus  
}  
#endif  
  
// White Points Defined in CIE 1931 Standard  
// Name      x      y      Color Temp  Comments  
// A         0.4476  0.4075  2854 K    Incandescent Light  
// B         0.3840  0.3516  4874 K    Direct Sunlight  
// C         0.3101  0.3162  6774 K    Indirect Sunlight
```

```
// D50    0.3457  0.3586  5000 K    Bright Incandescent Light
// D65    0.3127  0.3297  6504 K    "Natural" Daylight
// E      0.3333  0.3333  5500 K    Normalized Reference
// NTSC   0.310   0.316

// From ICC Profile Specification
// CIE Illuminant D50 [X=0.9642, Y=1.0000, Z=0.8249].

// rx, ry, gx, gy, bx, by, wx, wy
// 0.143555, 0.084961, 0.079102, 0.135742, 0.038086, 0.037109, 0.077148, 0.083984 // from PB G4 EDID data
// 0.67, 0.33, 0.21, 0.71, 0.14, 0.08; 0.310063, 0.316158 // 601
// 0.67, 0.33, 0.21, 0.71, 0.14, 0.08, 0.310063, 0.316158 // 709
// 0.64, 0.33, 0.29, 0.60, 0.15, 0.06, 0.312713, 0.329016 // D65
// 0.60, 0.33, 0.33, 0.54, 0.15, 0.12, 0.31, 0.33 // iMac LCD
// 0.63 0.35 0.30 0.59 0.14 0.11 0.31 0.33 // iMac LCD (newer)
// 0.64, 0.33, 0.28, 0.60, 0.14, 0.06, 0.28, 0.30 // eMac CRT
// 0.58, 0.34, 0.31, 0.53, 0.15, 0.11, 0.31, 0.33 // PBG4 400/500
// 0.60, 0.35, 0.32, 0.57, 0.15, 0.11, 0.31, 0.33 // PBG4 DVI
// 0.58 0.34 0.29 0.54 0.15 0.13 0.31 0.33 // iBook 12
// 0.58, 0.34, 0.34, 0.53, 0.16, 0.14, 0.31, 0.33 // iBook 14
// 0.64, 0.33, 0.30, 0.59, 0.15, 0.07, 0.31, 0.33 // Cinema HD Display 23
// 0.64, 0.34, 0.30, 0.60, 0.15, 0.10, 0.31, 0.33 // Cinema Display 22
// 0.63 0.35 0.30 0.57 0.14 0.09 0.31 0.33 // Studio Display 17 LCD
// 0.64 0.34 0.30 0.61 0.14 0.11 0.31 0.33 // Studio Display 15 LCD
// 0.63 0.34 0.28 0.60 0.15 0.07 0.28 0.30 // Studio Display 17 CRT
// 0.625, 0.340, 0.280, 0.595, 0.155, 0.070 // ColorSync 20-inch Display & AppleVision 850 Display.
The ColorSync 20-inch Display was formerly named the AppleVision 850 Display. The "AV" suffix is appended to the
monitor name as appropriate.

#endif /* __my_colorsync__ */
```

```
//  
//  
//  
#include "my_colorsenc.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
#define DEBUG  
#undef DEBUG  
  
typedef struct  
{  
    double mirek;    // temp (in microreciprocal kelvin)  
    double ut;       // u coord of intersection w/ blackbody locus  
    double vt;       // v coord of intersection w/ blackbody locus  
    double tt;       // slope of isotherm line  
} isotherm;  
  
// begin globals  
  
isotherm global_isothermdata[] =  
{  
    // Mirek, Ut, Vt, Tt  
    {0, 0.18006, 0.26352, -0.24341},  
    {10, 0.18066, 0.26589, -0.25479},  
    {20, 0.18133, 0.26846, -0.26876},  
    {30, 0.18208, 0.27119, -0.28539},  
    {40, 0.18293, 0.27407, -0.30470},  
    {50, 0.18388, 0.27709, -0.32675},  
    {60, 0.18494, 0.28021, -0.35156},  
    {70, 0.18611, 0.28342, -0.37915},  
    {80, 0.18740, 0.28668, -0.40955},  
    {90, 0.18880, 0.28997, -0.44278},  
    {100, 0.19032, 0.29326, -0.47888},  
    {125, 0.19462, 0.30141, -0.58204},  
    {150, 0.19962, 0.30921, -0.70471},  
    {175, 0.20525, 0.31647, -0.84901},  
    {200, 0.21142, 0.32312, -1.0182 },  
    {225, 0.21807, 0.32909, -1.2168 },  
    {250, 0.22511, 0.33439, -1.4512 },  
    {275, 0.23247, 0.33904, -1.7298 },  
    {300, 0.24010, 0.34308, -2.0637 },  
    {325, 0.24702, 0.34655, -2.4681 },  
    {350, 0.25591, 0.34951, -2.9641 },  
    {375, 0.26400, 0.35200, -3.5814 },  
    {400, 0.27218, 0.35407, -4.3633 },  
    {425, 0.28039, 0.35577, -5.3762 },  
    {450, 0.28863, 0.35714, -6.7262 },  
    {475, 0.29685, 0.35823, -8.5955 },  
    {500, 0.30505, 0.35907, -11.324 },  
    {525, 0.31320, 0.35968, -15.628 },  
    {550, 0.32129, 0.36011, -23.325 },  
    {575, 0.32931, 0.36038, -40.770 },  
    {600, 0.33724, 0.36051, -116.45 }  
};  
  
int global_niso = sizeof(global_isothermdata)/sizeof(isotherm);  
  
// CIE Color Matching Functions for wavelengths in 5 nm increments from 380 nm to 780 nm.  
double global_fColorMatch[][3]=  
{  
    // x_bar, y_bar, z_bar  
    {0.0014, 0.0000, 0.0065},  
    {0.0022, 0.0001, 0.0105},  
    {0.0042, 0.0001, 0.0201},  
    {0.0076, 0.0002, 0.0362},  
    {0.0143, 0.0004, 0.0679},  
    {0.0232, 0.0006, 0.1102},  
    {0.0435, 0.0012, 0.2074},  
    {0.0776, 0.0022, 0.3713},  
    {0.1344, 0.0040, 0.6456},  
    {0.2148, 0.0073, 1.0391},  
    {0.2839, 0.0116, 1.3856},  
    {0.3285, 0.0168, 1.6230},  
    {0.3483, 0.0230, 1.7471},  
    {0.3481, 0.0298, 1.7826},  
    {0.3362, 0.0380, 1.7721},  
    {0.3187, 0.0480, 1.7441},  
    {0.2908, 0.0600, 1.6692},  
    {0.2511, 0.0739, 1.5281},  
    {0.1954, 0.0910, 1.2876},  
    {0.1421, 0.1126, 1.0419},  
    {0.0956, 0.1390, 0.8130},  
    {0.0580, 0.1693, 0.6162},  
    {0.0320, 0.2080, 0.4652},  
}
```

```
{0.0147, 0.2586, 0.3533},
{0.0049, 0.3230, 0.2720},
{0.0024, 0.4073, 0.2123},
{0.0093, 0.5030, 0.1582},
{0.0291, 0.6082, 0.1117},
{0.0633, 0.7100, 0.0782},
{0.1096, 0.7932, 0.0573},
{0.1655, 0.8620, 0.0422},
{0.2257, 0.9149, 0.0298},
{0.2904, 0.9540, 0.0203},
{0.3597, 0.9803, 0.0134},
{0.4334, 0.9950, 0.0087},
{0.5121, 1.0000, 0.0057},
{0.5945, 0.9950, 0.0039},
{0.6784, 0.9786, 0.0027},
{0.7621, 0.9520, 0.0021},
{0.8425, 0.9154, 0.0018},
{0.9163, 0.8700, 0.0017},
{0.9786, 0.8163, 0.0014},
{1.0263, 0.7570, 0.0011},
{1.0567, 0.6949, 0.0010},
{1.0622, 0.6310, 0.0008},
{1.0456, 0.5668, 0.0006},
{1.0026, 0.5030, 0.0003},
{0.9384, 0.4412, 0.0002},
{0.8544, 0.3810, 0.0002},
{0.7514, 0.3210, 0.0001},
{0.6424, 0.2650, 0.0000},
{0.5419, 0.2170, 0.0000},
{0.4479, 0.1750, 0.0000},
{0.3608, 0.1382, 0.0000},
{0.2835, 0.1070, 0.0000},
{0.2187, 0.0816, 0.0000},
{0.1649, 0.0610, 0.0000},
{0.1212, 0.0446, 0.0000},
{0.0874, 0.0320, 0.0000},
{0.0636, 0.0232, 0.0000},
{0.0468, 0.0170, 0.0000},
{0.0329, 0.0119, 0.0000},
{0.0227, 0.0082, 0.0000},
{0.0158, 0.0057, 0.0000},
{0.0114, 0.0041, 0.0000},
{0.0081, 0.0029, 0.0000},
{0.0058, 0.0021, 0.0000},
{0.0041, 0.0015, 0.0000},
{0.0029, 0.0010, 0.0000},
{0.0020, 0.0007, 0.0000},
{0.0014, 0.0005, 0.0000},
{0.0010, 0.0004, 0.0000},
{0.0007, 0.0002, 0.0000},
{0.0005, 0.0002, 0.0000},
{0.0003, 0.0001, 0.0000},
{0.0002, 0.0001, 0.0000},
{0.0002, 0.0001, 0.0000},
{0.0001, 0.0000, 0.0000},
{0.0001, 0.0000, 0.0000},
{0.0001, 0.0000, 0.0000},
{0.0000, 0.0000, 0.0000}
};

// end globals

//
double do_max_of_3( double x, double y, double z )
{
double max;

max=x;

if(y>max)
max=y;

if(z>max)
max=z;

return max;
}

//
int do_calc_cct( double xs, double ys, double* t )
{
int j;
double us,vs;
double uj,vj,tj,di,dj,mi,mj;
double Tc;
```

```
/* convert (x,y) to CIE 1960 (u,v) */
us = (2*xs) / (-xs + 6*ys + 1.5);
vs = (3*ys) / (-xs + 6*ys + 1.5);

#ifdef DEBUG
printf("u = %lf, v = %lf\n", us, vs);
#endif

/* search for closest isotemp lines */
for(j=0; j<global_niso; j++)
{
    uj = global_isotempdata[j].ut;
    vj = global_isotempdata[j].vt;
    tj = global_isotempdata[j].tt;
    mj = global_isotempdata[j].mirek;

    /* dj = distance from (us,vs) to this isotemp line */
    dj = ((vs - vj) - tj * (us - uj)) / sqrt(1 + tj*tj);

#ifdef DEBUG
printf("j=%d d[j]=%lf T[j]=%lf\n",j,dj,1000000.0/mj);
#endif

    /* we stop when (di/dj) < 0.0, i.e. when distance changes */
    /* sign, because this means we have found isotemp lines */
    /* that "straddle" our point. */

    if ((j!=0) && (di/dj < 0.0))
    {
        Tc = 1000000.0 / (mi + (di / (di - dj)) * (mj - mi));
        break;
    }

    di = dj;
    mi = mj;
}

if (j == global_niso) return -1;

*t = Tc;
#ifdef DEBUG
printf("Tc = %lf\n",Tc);
#endif
return 0;
}

//
// XYZ VALUES FROM TEMPERATURE OF OBJECT
// A black body approximation is used where the temperature,
// T, is given in Kelvin. The XYZ values are determined by
// "integrating" the product of the wavelength distribution of
// energy and the XYZ functions for a uniform source.

void do_temp_to_XYZ( double temperature, double *X, double *Y, double *Z )
{
    double XX=0.0, YY=0.0, ZZ=0.0; /* initialize accumulators */
    double con, dis, wavelength, weight;
    short band, nbands=81;

    /* loop over wavelength bands */
    /* integration by trapezoid method */

    for(band=0; band<nbands; band++)
    {
        weight=1.0;

        if((band == 0)|| (band == nbands-1))
            weight = 0.5; /* properly weight end points */

        wavelength = 380.0 + (double)band*5.0; /* wavelength in nm */

        /* generate a black body spectrum */

        con=1240.0 / 8.617e-5;

        dis=3.74183e-16 * (1.0 / pow( wavelength, 5 ) ) / ( exp( con / ( wavelength * temperature ) ) - 1.0);

        /* simple integration over bands */
        XX=XX+weight*dis*global_fColorMatch[band][0];
        YY=YY+weight*dis*global_fColorMatch[band][1];
        ZZ=ZZ+weight*dis*global_fColorMatch[band][2];
    } /* end of 'band' loop */

    /* re-normalize the color scale */
}
```

```

    *X=XX/do_max_of_3(XX,YY,ZZ);
    *Y=YY/do_max_of_3(XX,YY,ZZ);
    *Z=ZZ/do_max_of_3(XX,YY,ZZ);
}

//

void do_temp_to_xy( double *x, double *y, double temp )
{
  double X,Y,Z,sum;

  do_temp_to_XYZ(temp, &X, &Y, &Z);

  sum = X + Y + Z;

  *x = X/sum;
  *y = Y/sum;
}

//

void do_xy_to_temp( double x, double y, double *temp )
{
  if(-1 == do_calc_cct(x, y, temp) )
    fprintf(stderr,"fail ");
}

//

void do_xy_To_XYZ( struct xyQuad *xy, struct XYZQuad *XYZ )
{
  float ar, ag, ab;
  float a[10][10];
  float b[10];
  float x,sum;
  int n,i,j,k;

  xy->red.z = 1.0 - xy->red.x - xy->red.y;
  xy->green.z = 1.0 - xy->green.x - xy->green.y;
  xy->blue.z = 1.0 - xy->blue.x - xy->blue.y;
  xy->white.z = 1.0 - xy->white.x - xy->white.y;

  // Unify XYZ->white.Y, then scale XYZ->white.X & XYZ->white.Z by relative amounts
  XYZ->white.X = xy->white.x / xy->white.y;
  XYZ->white.Y = 1.0; // xy->white.y / xy->white.y
  XYZ->white.Z = xy->white.z / xy->white.y;

  STATUS_VAR_PRINT("W = %12.4f",XYZ->white.X);
  STATUS_EXTRA_VAR_PRINT(", %12.4f",XYZ->white.Y);
  STATUS_EXTRA_VAR_PRINT(", %12.4f",XYZ->white.Z);

  // From Poynton
  // XYZ->white.Y = 1.0;
  // XYZ->white.X = ( xy->white.x / xy->white.y ) * XYZ->white.Y = xy->white.x / xy->white.y;
  // XYZ->white.Z = ( ( 1 - xy->white.x - xy->white.y ) / xy->white.y ) * XYZ->white.Y = ( xy->white.z / xy->white.y )
  // * XYZ->white.Y = xy->white.z / xy->white.y;

  // By rules of additivity
  // | xy->red.x  xy->green.x  xy->blue.x | * | ar | = XYZ->white.X = xy->white.x / xy->white.y;
  // | xy->red.y  xy->green.y  xy->blue.y |   | ag | = XYZ->white.Y = 1.0;
  // | xy->red.z  xy->green.z  xy->blue.z |   | ab | = XYZ->white.Z = xy->white.z / xy->white.y;

  // number of equations
  n = 3;

  // Matrix coefficients
  a[0][0] = xy->red.x;
  a[0][1] = xy->green.x;
  a[0][2] = xy->blue.x;

  a[1][0] = xy->red.y;
  a[1][1] = xy->green.y;
  a[1][2] = xy->blue.y;

  a[2][0] = xy->red.z;
  a[2][1] = xy->green.z;
  a[2][2] = xy->blue.z;

  // Right side vector
  b[0] = XYZ->white.X;
  b[1] = XYZ->white.Y;
  b[2] = XYZ->white.Z;

  // convert to upper triangular form
  for(k=0; k<n-1; k++)
  {
    if( fabs(a[k][k]) >= 1.e-6)

```

```
{
    for(i=k+1; i<n; i++)
    {
        x = a[i][k] / a[k][k];
        for(j=k+1; j<n; j++)
        {
            a[i][j] = a[i][j] - a[k][j] * x;
            a[i][j] -= (a[k][j] * x);
        }
        b[i] -= (b[k] * x);
    }
}
else
{
    printf( "zero pivot found in line:%d",k);
    return;
}
}

// back substitution
for(i=n-1; i>=0; i--)
{
    sum = b[i];
    if (i<n)
    {
        for(j=i+1; j<n; j++)
        {
            sum -= (a[i][j] * b[j]);
        }
        b[i] = sum / a[i][i];
    }
}

ar = b[0];
ag = b[1];
ab = b[2];

// STATUS_VAR_PRINT("Solution ar = %12.4f",ar);
// STATUS_EXTRA_VAR_PRINT(" , ag = %12.4f",ag);
// STATUS_EXTRA_VAR_PRINT(" , ab = %12.4f",ab);

XYZ->red.X = ar * xy->red.x;
XYZ->red.Y = ar * xy->red.y;
XYZ->red.Z = ar * xy->red.z;
STATUS_VAR_PRINT("R = %12.4f",XYZ->red.X);
STATUS_EXTRA_VAR_PRINT(" , %12.4f",XYZ->red.Y);
STATUS_EXTRA_VAR_PRINT(" , %12.4f",XYZ->red.Z);

XYZ->green.X = ag * xy->green.x;
XYZ->green.Y = ag * xy->green.y;
XYZ->green.Z = ag * xy->green.z;
STATUS_VAR_PRINT("G = %12.4f",XYZ->green.X);
STATUS_EXTRA_VAR_PRINT(" , %12.4f",XYZ->green.Y);
STATUS_EXTRA_VAR_PRINT(" , %12.4f",XYZ->green.Z);

XYZ->blue.X = ab * xy->blue.x;
XYZ->blue.Y = ab * xy->blue.y;
XYZ->blue.Z = ab * xy->blue.z;
STATUS_VAR_PRINT("B = %12.4f",XYZ->blue.X);
STATUS_EXTRA_VAR_PRINT(" , %12.4f",XYZ->blue.Y);
STATUS_EXTRA_VAR_PRINT(" , %12.4f",XYZ->blue.Z);

// (ar * xy->red.x) + (ag * xy->green.x) + (ab * xy->blue.x) = xy->white.x / xy->white.y;
// (ar * xy->red.y) + (ag * xy->green.y) + (ab * xy->blue.y) = xy->white.y / xy->white.y = 1.0;
// (ar * xy->red.z) + (ag * xy->green.z) + (ab * xy->blue.z) = xy->white.z / xy->white.y;
/*
// SUMn = XYZ->white.X + XYZ->white.Y + XYZ->white.Z;
xy->white.x = XYZ->white.X / ( XYZ->white.X + XYZ->white.Y + XYZ->white.Z );
xy->white.y = XYZ->white.Y / ( XYZ->white.X + XYZ->white.Y + XYZ->white.Z );
xy->white.z = XYZ->white.Z / ( XYZ->white.X + XYZ->white.Y + XYZ->white.Z );

XYZ->white.X = xy->white.x * ( XYZ->white.X + XYZ->white.Y + XYZ->white.Z );
XYZ->white.Y = xy->white.y * ( XYZ->white.X + XYZ->white.Y + XYZ->white.Z );
XYZ->white.Z = xy->white.z * ( XYZ->white.X + XYZ->white.Y + XYZ->white.Z );

// SUMr = XYZ->red.X + XYZ->red.Y + XYZ->red.Z;
xy->red.x = XYZ->red.X / ( XYZ->red.X + XYZ->red.Y + XYZ->red.Z );
xy->red.y = XYZ->red.Y / ( XYZ->red.X + XYZ->red.Y + XYZ->red.Z );
xy->red.z = XYZ->red.Z / ( XYZ->red.X + XYZ->red.Y + XYZ->red.Z );

XYZ->red.X = xy->red.x * ( XYZ->red.X + XYZ->red.Y + XYZ->red.Z );
XYZ->red.Y = xy->red.y * ( XYZ->red.X + XYZ->red.Y + XYZ->red.Z );
XYZ->red.Z = xy->red.z * ( XYZ->red.X + XYZ->red.Y + XYZ->red.Z );
```

```
// SUMg = XYZ->green.X + XYZ->green.Y + XYZ->green.Z;
xy->green.x = XYZ->green.X / ( XYZ->green.X + XYZ->green.Y + XYZ->green.Z );
xy->green.y = XYZ->green.Y / ( XYZ->green.X + XYZ->green.Y + XYZ->green.Z );
xy->green.z = XYZ->green.Z / ( XYZ->green.X + XYZ->green.Y + XYZ->green.Z );

// SUMb = XYZ->blue.X + XYZ->blue.Y + XYZ->blue.Z;
xy->blue.x = XYZ->blue.X / ( XYZ->blue.X + XYZ->blue.Y + XYZ->blue.Z );
xy->blue.y = XYZ->blue.Y / ( XYZ->blue.X + XYZ->blue.Y + XYZ->blue.Z );
xy->blue.z = XYZ->blue.Z / ( XYZ->blue.X + XYZ->blue.Y + XYZ->blue.Z );
*/
}
//

void do_XYZ_To_xy( struct XYZQuad *XYZ, struct xyQuad *xy )
{
float sum;

sum = XYZ->red.X + XYZ->red.Y + XYZ->red.Z;
xy->red.x = XYZ->red.X / sum;
xy->red.y = XYZ->red.Y / sum;
xy->red.z = XYZ->red.Z / sum;
// z = 1.0 - x - y;

sum = XYZ->green.X + XYZ->green.Y + XYZ->green.Z;
xy->green.x = XYZ->green.X / sum;
xy->green.y = XYZ->green.Y / sum;
xy->green.z = XYZ->green.Z / sum;

sum = XYZ->blue.X + XYZ->blue.Y + XYZ->blue.Z;
xy->blue.x = XYZ->blue.X / sum;
xy->blue.y = XYZ->blue.Y / sum;
xy->blue.z = XYZ->blue.Z / sum;

sum = XYZ->white.X + XYZ->white.Y + XYZ->white.Z;
xy->white.x = XYZ->white.X / sum;
xy->white.y = XYZ->white.Y / sum;
xy->white.z = XYZ->white.Z / sum;
}

//
/*
void main()
{
double x,y,temp,temp2;

for(temp = 1000.0; temp <= 30000.0; temp += 100.0)
{
do_temp_to_xy(&x, &y, temp);
do_xy_to_temp(x, y, &temp2);

fprintf(stderr,"temp_in = %f x = %f y = %f temp_out = %f error is
%f%%\n",temp,x,y,temp2,(temp2-temp)/temp*100.0);
}
}
*/
```



```
//  
//  
//  
#include "my_controls.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
//  
//  
//  
void do_Dump_Control_Hierarchy ( WindowPtr window_ptr )  
{  
    //StandardFileReply    the_reply;  
    NavReplyRecord        the_reply;  
    FSSpec                the_file;  
  
    // This is needed when using the floating window routines.  
    // DeactivateFloatersAndFirstDocumentWindow();  
  
    // StandardPutFile( "\pSave File As:", "\pUntitled", &the_reply );  
    do_Nav_Put_File( "\pSave File As:", "\pUntitled", 'TEXT', 'CWIE', NULL, &the_file, &the_reply);  
  
    // Once we get control back, we reactivate our floating windows.  
    // ActivateFloatersAndFirstDocumentWindow();  
  
    // if ( the_reply.sfGood != false )  
    // if ( the_reply.validRecord )  
    // {  
    //     if ( the_reply.sfReplacing == false )  
    //     if ( the_reply.replacing == false )  
    //     {  
    //         FSpCreate( &the_reply.sfFile, 'CWIE', 'TEXT', smSystemScript );  
    //         FSpCreate( &the_file, 'CWIE', 'TEXT', smSystemScript );  
    //     }  
  
    //     DumpControlHierarchy ( window_ptr, &the_reply.sfFile );  
    //     DumpControlHierarchy ( window_ptr, &the_file );  
    // }  
  
    NavDisposeReply( &the_reply );  
}  
  
//  
//  
//  
void do_Print_Supported_Control_Features ( ControlHandle whichControl )  
{  
    long                control_ref_con;  
    //short            out_data;  
    UInt32            features;  
    OSErr            err;  
  
    control_ref_con = GetControlReference ( whichControl );  
  
    DEBUG_VAR_PRINT( "Control hit was: %d\n", control_ref_con );  
  
    // err = GetControlData ( whichControl, kControlNoPart, kControlMsgGetFeatures, sizeof(UInt32), (Ptr)&features,  
    // nil );  
    err = GetControlFeatures ( whichControl, &features );  
  
    if ( err == errMsgNotSupported )  
    {  
        DEBUG_PRINT( "Control does not support appearance-compliant features.\n" );  
        return;  
    }  
  
    if ( features & kControlSupportsGhosting )  
    {  
        DEBUG_PRINT( "Control supports ghosting.\n" );  
    }  
    else  
    {  
        DEBUG_PRINT( "Control does not support ghosting.\n" );  
    }  
  
    if ( features & kControlSupportsEmbedding )  
    {  
        DEBUG_PRINT( "Control supports embedding.\n" );  
    }  
    else  
    {  
        DEBUG_PRINT( "Control does not support embedding.\n" );  
    }  
  
    if ( features & kControlSupportsFocus )  
    {
```

```
        DEBUG_PRINT( "Control supports focus.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not support focus.\n" );
    }

    if ( features & kControlWantsIdle )
    {
        DEBUG_PRINT( "Control wants idle.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not want idle.\n" );
    }

    if ( features & kControlWantsActivate )
    {
        DEBUG_PRINT( "Control wants activate.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not want activate.\n" );
    }

    if ( features & kControlHandlesTracking )
    {
        DEBUG_PRINT( "Control handles tracking.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not handle tracking.\n" );
    }

    if ( features & kControlSupportsDataAccess )
    {
        DEBUG_PRINT( "Control supports data access.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not support data access.\n" );
    }

    if ( features & kControlHasSpecialBackground )
    {
        DEBUG_PRINT( "Control has special background.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not have special background.\n" );
    }

    if ( features & kControlGetsFocusOnClick )
    {
        DEBUG_PRINT( "Control gets focus on click.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not get focus on click.\n" );
    }

    if ( features & kControlSupportsCalcBestRect )
    {
        DEBUG_PRINT( "Control supports calc best rect.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not support calc best rect.\n" );
    }

    if ( features & kControlSupportsLiveFeedback )
    {
        DEBUG_PRINT( "Control supports live feedback.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not support live feedback.\n" );
    }

    if ( features & kControlHasRadioBehavior )
    {
        DEBUG_PRINT( "Control has radio behavior.\n" );
    }
    else
    {
        DEBUG_PRINT( "Control does not have radio behavior.\n" );
    }
}
```

```
//  
//  
//  
#ifndef __my_dialogs__  
#define __my_dialogs__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <ControlDefinitions.h>  
#include <TextUtils.h>  
#include <Dialogs.h>  
#include <Scrap.h>  
#include <Sound.h>  
#include <Lists.h>  
#include <stdlib.h>  
#endif  
#endif  
  
#include "my_macros.h"  
#include "my_strings.h"  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
OSStatus do_Get_Value_Of_DItem ( DialogRef, int, SInt16 * );  
OSStatus do_Set_Value_Of_DItem ( DialogRef, int, short );  
  
OSStatus do_Set_Value_Of_DItem_As_Clippled ( DialogRef, int, short );  
OSStatus do_Set_Value_Of_DItem_As_Boolean ( DialogRef, int, short );  
OSStatus do_Set_Value_Of_DItem_As_Mixed ( DialogRef, int, short );  
  
OSStatus do_Get_Min_Value_Of_DItem ( DialogRef, int, SInt16 * );  
OSStatus do_Set_Min_Value_Of_DItem ( DialogRef, int, SInt16 );  
  
OSStatus do_Get_Max_Value_Of_DItem ( DialogRef, int, SInt16 * );  
OSStatus do_Set_Max_Value_Of_DItem ( DialogRef, int, SInt16 );  
  
OSStatus do_Get_Text_Of_DItem_As_PString ( DialogRef, int, unsigned char * );  
OSStatus do_Get_Text_Of_DItem_As_CString ( DialogRef, int, char *, unsigned short );  
OSStatus do_Set_Text_Of_DItem_As_PString ( DialogRef, int, const unsigned char *, Boolean );  
OSStatus do_Set_Text_Of_DItem_As_CString ( DialogRef, int, const char *, Boolean );  
  
OSStatus do_Get_Text_Of_DItem_As_Float ( DialogRef, int, float * );  
OSStatus do_Set_Text_Of_DItem_As_Float ( DialogRef, int, float, Boolean );  
  
OSStatus do_Get_Text_Of_DItem_As_Int ( DialogRef, int, long * );  
OSStatus do_Set_Text_Of_DItem_As_Int ( DialogRef, int, long, Boolean );  
  
OSStatus do_Get_Text_Of_DItem_As_Fixed ( DialogRef, int, Fixed * );  
OSStatus do_Set_Text_Of_DItem_As_Fixed ( DialogRef, int, Fixed, Boolean );  
  
OSStatus do_Set_Text_Style_Of_DItem ( DialogRef, int, short, short, short );  
OSStatus do_Get_Title_Of_DItem ( DialogRef, int, unsigned char * );  
OSStatus do_Set_Title_Of_DItem ( DialogRef, int, const unsigned char * );  
OSStatus do_Set_Bevel_Button_Text_Placement ( DialogRef, int, ControlButtonTextPlacement, short );  
OSStatus do_Set_Bevel_Button_Text_Alignment ( DialogRef, int, ControlButtonTextAlignment, short );  
OSStatus do_Set_Bevel_Button_Graphic_Alignment ( DialogRef, int, ControlButtonGraphicAlignment, short, short );  
OSStatus do_Get_Bevel_Button_Content_Info ( DialogRef, int, const ControlButtonContentInfoPtr );  
OSStatus do_Set_Bevel_Button_Content_Info ( DialogRef, int, ControlButtonContentInfoPtr );  
  
OSStatus do_Get_List_Box_List_Handle ( ControlHandle, ListHandle * );  
OSStatus do_Set_List_Box_Key_Filter ( ControlHandle, ControlKeyFilterUPP );  
OSStatus do_Set_Key_Filter_Of_DItem ( DialogRef, int, ControlKeyFilterUPP );  
  
void do_Set_Control_Ref_Of_DItem ( DialogRef, int, long );  
void do_Set_Control_Action_Of_DItem ( DialogRef, int, ControlActionUPP );  
OSStatus do_Set_Control_Draw_Proc_Of_DItem ( DialogRef, int, ControlUserPaneDrawUPP );  
OSStatus do_Set_Control_Hit_Test_Proc_Of_DItem ( DialogRef, int, ControlUserPaneHitTestUPP );  
  
void do_Activate_DItem ( DialogRef, int, Boolean );
```

```
void      do_Activate_Control ( ControlHandle, Boolean );
Boolean   do_Is_DItem_Active ( DialogRef, int );

void      do_Show_DItem ( DialogRef, int, Boolean );
void      do_Show_Control ( ControlHandle, Boolean );
Boolean   do_Is_DItem_Visible ( DialogRef, int );

OSStatus  do_Draw_One_Control_As_DItem ( DialogRef, int );

OSStatus  do_Get_Progress_State_Of_DItem ( DialogRef, int, Boolean * );
OSStatus  do_Set_Progress_State_Of_DItem ( DialogRef, int, Boolean );

OSStatus  do_Get_Progress_State_Of_Control ( ControlHandle, Boolean * );
OSStatus  do_Set_Progress_State_Of_Control ( ControlHandle, Boolean );

OSStatus  do_Get_Tab_Content_Rect_Of_DItem ( DialogRef, int, Rect * );
OSStatus  do_Get_Tab_Content_Rect_Of_Control ( ControlHandle, Rect * );

OSStatus  do_Enable_Tab_Of_DItem ( DialogRef, int, SInt16, Boolean );
OSStatus  do_Enable_Tab_Of_Control ( ControlHandle, SInt16, Boolean );

short     do_Get_Keyboard_Focus_As_DItem ( DialogRef );
OSStatus  do_Set_Keyboard_Focus_As_DItem ( DialogRef, int );

OSStatus  do_Get_Popup_Button_Menu_Handle_Of_DItem ( DialogRef, int, MenuHandle* );
OSStatus  do_Get_Popup_Button_Menu_ID_Of_DItem ( DialogRef, int, short * );

Boolean   do_Check_Scrap_Is_Only_Digits ( void );
short     do_Get_Selection_Length ( DialogRef );
short     do_Get_Current_Edit_Field ( DialogRef );

#ifdef __cplusplus
}
#endif

#endif /* __my_dialogs__ */
```

```

// _____ ©1998-2001 bergdesign inc.
// _____

#include "my_dialogs.h"

// _____
// All of the following functions assume that there is an
// embedding hierarchy present in the dialog. Since we're
// fully adopting the Appearance Manager, this is the price
// we'll have to pay.
// _____

OSStatus do_Get_Value_Of_DItem ( DialogRef dlog, int item, Sint16 *value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
        *value = GetControlValue ( control );

    return( err );
}

// _____

OSStatus do_Set_Value_Of_DItem ( DialogRef dlog, int item, short value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
        SetControlValue ( control, value );

    return( err );
}

// _____

OSStatus do_Set_Value_Of_DItem_As_Clipped ( DialogRef dlog, int item, short value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;
    short         min, max;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
    {
        // this will clip the value so that it is within the allowed limits
        min = GetControlMinimum ( control );
        max = GetControlMaximum ( control );
        value = MIN ( MAX ( min, value ), max );

        SetControlValue ( control, value );
    }

    return( err );
}

// _____

OSStatus do_Set_Value_Of_DItem_As_Boolean ( DialogRef dlog, int item, short value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    // "val != 0" will evaluate to a Boolean 0 or 1
    if ( ( NULL != control ) && ( noErr == err ) )
        SetControlValue ( control, (value != 0) );

    return( err );
}

// _____

OSStatus do_Set_Value_Of_DItem_As_Mixed ( DialogRef dlog, int item, short value )
{

```

```
OSStatus      err = noErr;
ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

// This will clip the value so that it is within the allowed limits
// for a mixed value control, like an Appearance savvy radio button
if ( ( NULL != control ) && ( noErr == err ) )
{
    value = MIN ( MAX ( 0, value ), 2 );

    SetControlValue ( control, value );
}

return( err );
}

//
#pragma mark -

//
OSStatus do_Get_Min_Value_Of_DItem ( DialogRef dlog, int item, SInt16 *value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
        *value = GetControlMinimum ( control );

    return( err );
}

//
OSStatus do_Set_Min_Value_Of_DItem ( DialogRef dlog, int item, SInt16 value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
        SetControlMinimum ( control, value );

    return( err );
}

//
OSStatus do_Get_Max_Value_Of_DItem ( DialogRef dlog, int item, SInt16 *value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
        *value = GetControlMaximum ( control );

    return( err );
}

//
OSStatus do_Set_Max_Value_Of_DItem ( DialogRef dlog, int item, SInt16 value )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
        SetControlMaximum ( control, value );

    return( err );
}

//
#pragma mark -

//
// do_Get_Text_Of_DItem_As_PString() will get the text in an
// editText or staticText control. The short/long/float
```

```
// routines that follow use this routine to fetch the item's
// contents.

OSStatus do_Get_Text_Of_DItem_As_PString ( DialogRef dlog, int item, unsigned char *string )
{
    Size          actual_size;
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    if ( string == NULL )
        return ( paramErr );

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( NULL == control || noErr != err )
        return ( err );

    // This routine can be used for both static and edit text items.
    // Since kControlEditTextTextTag = kControlStaticTextTextTag, it works for both.
    err = GetControlData ( control, 0, kControlEditTextTextTag, 255, (Ptr)(string + 1), &actual_size );

    // If everything went ok, we truncate the string to be safe.
    if ( err == noErr )
        string[0] = MIN ( 255, actual_size );
    else
        string[0] = 0;

    return ( err );
}

//
OSStatus do_Get_Text_Of_DItem_As_CString ( DialogRef dlog, int item, char *c_string, unsigned short length )
{
    OSStatus      err = noErr;
    unsigned char string[256];

    err = do_Get_Text_Of_DItem_As_PString ( dlog, item, string );

    if ( err == noErr )
    {
        do_p2c_str ( string );
        length = MIN ( length, 255 );
        strncpy ( c_string, (char *)string, length );
        c_string[length] = '\0';
    }

    return ( err );
}

//
// The short/long/float routines that follow use
// this routine to set the item's contents.

OSStatus do_Set_Text_Of_DItem_As_PString ( DialogRef dlog, int item, const unsigned char *string, Boolean
do_select )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    if ( string == NULL )
        return( paramErr );

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( NULL == control || noErr != err )
        return( err );

    // This routine can be used for both static and edit text items.
    // Since kControlEditTextTextTag = kControlStaticTextTextTag, it works for both.
    err = SetControlData ( control, 0, kControlEditTextTextTag, string[0], (Ptr)(string+1) );

    // If there was no error, the control was the proper type.
    if ( err == noErr )
    {
        ControlEditTextSelectionRec  selection;

        // If desired, we hilite the text in the item.
        if ( do_select )
            selection.selStart = 0;
        else
            selection.selStart = 32767;

        selection.selEnd = 32767;

        err = SetControlData ( control, 0, kControlEditTextSelectionTag, sizeof( selection ), (Ptr)&selection );
    }
}
```



```
        DrawOneControl ( control );

    return ( err );
}

// _____

OSStatus do_Set_Text_Of_DItem_As_CString ( DialogRef dlog, int item, const char *string, Boolean do_select )
{
    OSStatus          err = noErr;
    unsigned char      p_string[256];

    strncpy ( (char *)p_string, string, 255 );
    p_string[255] = '\0';
    do_c2p_str ( (char *)p_string );

    err = do_Set_Text_Of_DItem_As_PString ( dlog, item, p_string, do_select );

    return ( err );
}

// _____

OSStatus do_Get_Text_Of_DItem_As_Float ( DialogRef dlog, int item, float *value )
{
    OSStatus          err = noErr;
    int               count;
    unsigned char      string[256];

    *value = NULL;
    err = do_Get_Text_Of_DItem_As_PString ( dlog, item, string );

    if ( err == noErr )
    {
        do_p2c_str ( string );
        count = sscanf ( (char *)string, "%f", value );
        if ( count != 1 )
            return ( paramErr );
    }

    return ( err );
}

// _____

OSStatus do_Set_Text_Of_DItem_As_Float ( DialogRef dlog, int item, float value, Boolean do_select )
{
    OSStatus          err = noErr;
    char               string[256];

    sprintf ( string, "%5.3f", value );
    do_c2p_str ( string );
    err = do_Set_Text_Of_DItem_As_PString ( dlog, item, (unsigned char *)string, do_select );

    return ( err );
}

// _____

OSStatus do_Get_Text_Of_DItem_As_Int ( DialogRef dlog, int item, long *value )
{
    OSStatus          err = noErr;
    unsigned char      string[256];

    *value = NULL;
    err = do_Get_Text_Of_DItem_As_PString ( dlog, item, string );

    if ( err == noErr )
        StringToNum ( string, value );

    return ( err );
}

// _____

OSStatus do_Set_Text_Of_DItem_As_Int ( DialogRef dlog, int item, long value, Boolean do_select )
{
    OSStatus          err = noErr;
    unsigned char      string[256];

    NumToString ( value, string );
    err = do_Set_Text_Of_DItem_As_PString ( dlog, item, string, do_select );

    return ( err );
}

// _____
```

```
OSStatus do_Get_Text_Of_DItem_As_Fixed ( DialogRef dlog, int item, Fixed *value )
{
    OSStatus      err = noErr;
    int           count;
    float         float_value;
    unsigned char string[256];

    *value = NULL;
    err = do_Get_Text_Of_DItem_As_PString ( dlog, item, string );

    if ( err == noErr )
    {
        do_p2c_str ( string );
        count = sscanf ( (char *)string, "%f", &float_value );
        if ( count != 1 )
            return 0;
        *value = FloatToFixed ( float_value );
    }

    return ( err );
}

//

OSStatus do_Set_Text_Of_DItem_As_Fixed ( DialogRef dlog, int item, Fixed value, Boolean do_select )
{
    OSStatus      err = noErr;
    float         float_value;
    unsigned char string[256];

    float_value = FixedToFloat ( value );
    sprintf ( (char *)string, "%1.3f", float_value );
    err = do_Set_Text_Of_DItem_As_PString ( dlog, item, string, do_select );

    return ( err );
}

//

#pragma mark -

//

short do_Get_Keyboard_Focus_As_DItem ( DialogRef dlog )
{
    OSStatus      err = noErr;
    ControlHandle control = NULL;
    ControlHandle counter_control = NULL;
    short         item = -1;
    short         counter, total_num_items;

    err = GetKeyboardFocus ( GetDialogWindow(dlog), &control );

    if ( err == errNoRootControl )
    {
        return ( item );
    }
    else
    {
        total_num_items = CountDITL ( dlog );
        for ( counter = 1; counter <= total_num_items; counter++ )
        {
            GetDialogItemAsControl ( dlog, counter, &counter_control );

            if ( counter_control == control )
            {
                item = counter;
                break;
            }
        }
    }

    return ( item );
}

//

OSStatus do_Set_Keyboard_Focus_As_DItem ( DialogRef dlog, int item )
{
    OSStatus      err = noErr;
    ControlHandle control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( ( NULL != control ) && ( noErr == err ) )
        err = SetKeyboardFocus ( GetDialogWindow(dlog), control, kControlEditTextPart );

    return ( err );
}
```

```
}  
  
//  
  
#pragma mark -  
  
//  
  
OSStatus do_Set_Text_Style_Of_DItem ( DialogRef dlog, int item, short font, short size, short style, short just )  
{  
    OSStatus          err = noErr;  
    ControlHandle      control = NULL;  
    ControlFontStyleRec font_style_rec;  
  
    GetDialogItemAsControl ( dlog, item, &control );  
  
    if ( control )  
    {  
        font_style_rec.flags = 0;  
  
        if ( font )  
            font_style_rec.flags += kControlUseFontMask;  
  
        if ( size )  
            font_style_rec.flags += kControlUseSizeMask;  
  
        if ( style )  
            font_style_rec.flags += kControlUseFaceMask;  
  
        if ( just )  
            font_style_rec.flags += kControlUseJustMask;  
  
        font_style_rec.font = font;  
        font_style_rec.size = size;  
        font_style_rec.style = style;  
        font_style_rec.just = just;  
  
        err = SetControlData ( control, 0, kControlFontStyleTag, sizeof( ControlFontStyleRec ), (Ptr)&font_style_rec );  
    }  
    else  
    {  
        err = paramErr;  
    }  
  
    return ( err );  
}  
  
//  
  
OSStatus do_Get_Title_Of_DItem ( DialogRef dlog, int item, unsigned char *string )  
{  
    OSStatus          err = noErr;  
    ControlHandle      control = NULL;  
  
    err = GetDialogItemAsControl ( dlog, item, &control );  
  
    if ( ( NULL != control ) && ( noErr == err ) )  
        GetControlTitle ( control, string );  
  
    return( err );  
}  
  
//  
  
OSStatus do_Set_Title_Of_DItem ( DialogRef dlog, int item, const unsigned char *string )  
{  
    OSStatus          err = noErr;  
    ControlHandle      control = NULL;  
  
    err = GetDialogItemAsControl ( dlog, item, &control );  
  
    if ( ( NULL != control ) && ( noErr == err ) )  
        SetControlTitle ( control, string );  
  
    return( err );  
}  
  
//  
  
OSStatus do_Set_Bevel_Button_Text_Placement ( DialogRef dlog, int item, ControlButtonTextPlacement placement,  
short hOffset )  
{  
    OSStatus          err = noErr;  
    ControlHandle      control = NULL;  
  
    GetDialogItemAsControl ( dlog, item, &control );
```

```
    if ( !control )
        return paramErr;

    err = SetControlData ( control, 0, kControlBevelButtonTextPlaceTag, sizeof( ControlButtonTextPlacement ),
(Ptr)&placement );

    if ( !err )
        err = SetControlData ( control, 0, kControlBevelButtonTextOffsetTag, sizeof( short ), (Ptr)&hOffset );

    return err;
}

// _____

OSStatus do_Set_Bevel_Button_Text_Alignment ( DialogRef dlog, int item, ControlButtonTextAlignment align, short
hOffset )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( !control )
        return paramErr;

    err = SetControlData ( control, 0, kControlBevelButtonTextAlignTag, sizeof( ControlButtonTextAlignment ),
(Ptr)&align );

    if ( !err )
        err = SetControlData ( control, 0, kControlBevelButtonTextOffsetTag, sizeof( short ), (Ptr)&hOffset );

    return err;
}

// _____

OSStatus do_Set_Bevel_Button_Graphic_Alignment ( DialogRef dlog, int item, ControlButtonGraphicAlignment align,
short hOffset, short vOffset )
{
    OSStatus      err = noErr;
    Point          offset;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( !control )
        return paramErr;

    err = SetControlData ( control, 0, kControlBevelButtonGraphicAlignTag, sizeof( ControlButtonGraphicAlignment ),
(Ptr)&align );

    if ( !err )
    {
        offset.h = hOffset;
        offset.v = vOffset;

        err = SetControlData ( control, 0, kControlBevelButtonGraphicOffsetTag, sizeof( Point ), (Ptr)&offset );
    }

    return err;
}

// _____

OSStatus do_Get_Bevel_Button_Content_Info ( DialogRef dlog, int item, const ControlButtonContentInfoPtr info )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( !control )
        return paramErr;

    err = GetControlData ( control, 0, kControlBevelButtonContentTag, sizeof( ControlButtonContentInfo ),
(Ptr)info, NULL );

    return err;
}

// _____

OSStatus do_Set_Bevel_Button_Content_Info ( DialogRef dlog, int item, ControlButtonContentInfoPtr info )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );
```

```
    if ( !control )
        return paramErr;

    err = SetControlData ( control, 0, kControlBevelButtonContentTag, sizeof( ControlButtonContentInfo ), (Ptr)info
);
    return err;
}

// _____

#pragma mark -

// _____

void do_Set_Control_Ref_Of_DItem ( DialogRef dlog, int item, long ref_con )
{
    ControlHandle    control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
    {
        SetControlReference ( control, ref_con );
    }
}

// _____

void do_Set_Control_Action_Of_DItem ( DialogRef dlog, int item, ControlActionUPP action )
{
    ControlHandle    control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
        SetControlAction ( control, action );
}

// _____

OSStatus do_Set_Control_Draw_Proc_Of_DItem ( DialogRef dlog, int item, ControlUserPaneDrawUPP draw_proc )
{
    OSStatus        err = noErr;
    ControlHandle    control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( !err && control )
        err = SetControlData ( control, 0, kControlUserPaneDrawProcTag, sizeof( ControlUserPaneDrawUPP ),
(Ptr)&draw_proc );

    return ( err );
}

// _____

OSStatus do_Set_Control_Hit_Test_Proc_Of_DItem ( DialogRef dlog, int item, ControlUserPaneHitTestUPP hit_test_proc )
{
    OSStatus        err = noErr;
    ControlHandle    control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( !err && control )
        err = SetControlData ( control, 0, kControlUserPaneHitTestProcTag, sizeof( ControlUserPaneHitTestUPP ),
(Ptr)&hit_test_proc );

    return ( err );
}

// _____

#pragma mark -

// _____

void do_Activate_DItem ( DialogRef dlog, int item, Boolean activate )
{
    ControlHandle    control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    do_Activate_Control ( control, activate );
}
```

```
//  
  
void do_Activate_Control ( ControlHandle control, Boolean activate )  
{  
    if ( control )  
    {  
        Boolean active = IsControlActive ( control );  
  
        if ( activate )  
        {  
            if ( !active )  
                ActivateControl ( control );  
        }  
        else  
        {  
            if ( active )  
                DeactivateControl ( control );  
        }  
    }  
}
```

```
//  
  
Boolean do_Is_DItem_Active ( DialogRef dlog, int item )  
{  
    ControlHandle    control = NULL;  
    Boolean          active = false;  
  
    GetDialogItemAsControl ( dlog, item, &control );  
  
    if ( control )  
        active = IsControlActive ( control );  
  
    return ( active );  
}
```

```
//  
  
void do_Show_DItem ( DialogRef dlog, int item, Boolean show )  
{  
    ControlHandle    control = NULL;  
  
    GetDialogItemAsControl ( dlog, item, &control );  
  
    do_Show_Control ( control, show );  
}
```

```
//  
  
void do_Show_Control ( ControlHandle control, Boolean show )  
{  
    if ( control )  
    {  
        Boolean visible = IsControlVisible ( control );  
  
        if ( show )  
        {  
            if ( !visible )  
                ShowControl ( control );  
        }  
        else  
        {  
            if ( visible )  
                HideControl ( control );  
        }  
    }  
}
```

```
//  
  
Boolean do_Is_DItem_Visible ( DialogRef dlog, int item )  
{  
    ControlHandle    control = NULL;  
    Boolean          visible = false;  
  
    GetDialogItemAsControl ( dlog, item, &control );  
  
    if ( control )  
        visible = IsControlVisible ( control );  
  
    return ( visible );  
}
```

```
//  
  
OSStatus do_Draw_One_Control_As_DItem ( DialogRef dlog, int item )  
{  
    OSStatus          err = noErr;
```

```
ControlHandle    control = NULL;

    err = GetDialogItemAsControl ( dlog, item, &control );

    if ( !err && control )
        DrawControl ( control );

    return ( err );
}

// _____

#pragma mark -

// _____

OSStatus do_Set_Progress_State_Of_DItem ( DialogRef dlog, int item, Boolean is_determinate )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
        err = do_Set_Progress_State_Of_Control ( control, is_determinate );
    else
        err = paramErr;

    return err;
}

// _____

OSStatus do_Set_Progress_State_Of_Control ( ControlHandle control, Boolean is_determinate )
{
    OSStatus      err = noErr;
    Boolean        state;

    if ( control == NULL )
        return paramErr;

    state = !is_determinate;
    err = SetControlData ( control, 0, kControlProgressBarIndeterminateTag, sizeof( state ), (Ptr)&state );

    return err;
}

// _____

OSStatus do_Get_Progress_State_Of_DItem ( DialogRef dlog, int item, Boolean *is_determinate )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
        err = do_Get_Progress_State_Of_Control ( control, is_determinate );
    else
        err = paramErr;

    return err;
}

// _____

OSStatus do_Get_Progress_State_Of_Control ( ControlHandle control, Boolean *is_determinate )
{
    Size          actualSize;
    OSStatus      err = noErr;
    Boolean        temp;

    if ( control == NULL )
        return paramErr;

    if ( is_determinate == NULL )
        return paramErr;

    err = GetControlData ( control, 0, kControlListBoxListHandleTag, sizeof( temp ), (Ptr)&temp, &actualSize );

    if ( err == noErr )
        *is_determinate = !temp;

    return err;
}

// _____
```

#pragma mark -

```
//
OSStatus do_Get_Tab_Content_Rect_Of_DItem ( DialogRef dlog, int item, Rect *content_rect )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
        err = do_Get_Tab_Content_Rect_Of_Control ( control, content_rect );
    else
        err = paramErr;

    return err;
}
//
```

```
OSStatus do_Get_Tab_Content_Rect_Of_Control ( ControlHandle control, Rect *content_rect )
{
    OSStatus      err;

    if ( control == NULL )
        return paramErr;

    err = GetControlData ( control, 0, kControlTabContentRectTag, sizeof( Rect ), (Ptr)content_rect, NULL );
    return err;
}
//
```

```
OSStatus do_Enable_Tab_Of_DItem ( DialogRef dlog, int item, Sint16 tab_to_hilite, Boolean enable )
{
    OSStatus      err = noErr;
    ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
        err = do_Enable_Tab_Of_Control ( control, tab_to_hilite, enable );
    else
        err = paramErr;

    return err;
}
//
```

```
OSStatus do_Enable_Tab_Of_Control ( ControlHandle control, Sint16 tab_to_hilite, Boolean enable )
{
    OSStatus      err;

    if ( control == NULL )
        return paramErr;

    err = SetControlData( control, tab_to_hilite, kControlTabEnabledFlagTag, sizeof( Boolean ), (Ptr)&enable );
    return err;
}
//
```

#pragma mark -

```
//
OSStatus do_Get_List_Box_List_Handle ( ControlHandle control, ListHandle* list )
{
    Size          actualSize;
    OSStatus      err;

    if ( control == nil )
        return paramErr;

    if ( list == nil )
        return paramErr;

    err = GetControlData ( control, 0, kControlListBoxListHandleTag, sizeof( ListHandle ),
        (Ptr)list, &actualSize );

    return err;
}
//
```

```
OSStatus do_Set_List_Box_Key_Filter ( ControlHandle control, ControlKeyFilterUPP filter )
```



```
{
OSStatus      err;

    if ( control == nil )
        return paramErr;

    if ( filter == nil )
        return paramErr;

    err = SetControlData ( control, 0, kControlKeyFilterTag, sizeof( filter ), (Ptr)&filter );

    return err;
}

//_____

OSStatus do_Set_Key_Filter_Of_DItem ( DialogRef dlog, int item, ControlKeyFilterUPP filter )
{
OSStatus      err = noErr;
ControlHandle  control = NULL;

    if ( filter == nil )
        return paramErr;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control == nil )
        return paramErr;

    err = SetControlData ( control, 0, kControlKeyFilterTag, sizeof( filter ), (Ptr)&filter );

    return err;
}

//_____

#pragma mark -

//_____

OSStatus do_Get_Popup_Button_Menu_Handle_Of_DItem ( DialogRef dlog, int item, MenuHandle *menu_handle )
{
OSStatus      err = noErr;
ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
        err = GetControlData ( control, kControlEntireControl, kControlPopupMenuHandleTag, sizeof( MenuHandle ), (Ptr)menu_handle, nil );
    else
        err = paramErr;

    return err;
}

//_____

OSStatus do_Get_Popup_Button_Menu_ID_Of_DItem ( DialogRef dlog, int item, short *menu_id )
{
OSStatus      err = noErr;
ControlHandle  control = NULL;

    GetDialogItemAsControl ( dlog, item, &control );

    if ( control )
        err = GetControlData ( control, kControlEntireControl, kControlPopupMenuIDTag, sizeof( short ), (Ptr)menu_id, nil );
    else
        err = paramErr;

    return err;
}

//_____

#pragma mark -

//_____

Boolean do_Check_Scrap_Is_Only_Digits ( void )
{
Ptr            text_ptr = NULL;
Boolean        only_digits = true;
unsigned short i;
ScrapRef       the_scrap;
Size           the_size;
}
```

```
GetCurrentScrap( &the_scrap );
GetScrapFlavorSize ( the_scrap, 'TEXT', &the_size );

if ( the_size <= 0 )
    return ( false );

text_ptr = NewPtr ( the_size );
if ( text_ptr != NULL )
{
    GetScrapFlavorData( the_scrap, 'TEXT', &the_size, text_ptr );

    for ( i = 0; i < the_size; i++ )
    {
        switch ( text_ptr[i] )
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
            case '-': break;

            default:    only_digits = false;
                       break;
        }
    }

    DisposePtr ( text_ptr );
}

return ( only_digits );
}

//
short do_Get_Selection_Length ( DialogRef dlog )
{
    TEHandle    text_edit_handle;

    // text_edit_handle = ((DialogPeek)dlog)->textH;
    text_edit_handle = GetDialogTextEditHandle(dlog);

    return( (**text_edit_handle).selEnd - (**text_edit_handle).selStart );
}

//
/*
short do_Get_Current_Edit_Field ( DialogRef dlog )
{
    return( ((DialogPeek)dlog)->editField + 1 );
}
*/
```

```
//  
//  
//  
#ifndef __my_displays__  
#define __my_displays__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Dialogs.h>  
#include <Devices.h>  
#include <Displays.h>  
#include <Errors.h>  
#include <FixMath.h>  
#include <fp.h>  
#include <Gestalt.h>  
#include <Memory.h>  
#include <Palettes.h>  
#include <PLStringFuncs.h>  
#include <QuickDraw.h>  
#include <ROMDefs.h>  
#include <Slots.h>  
#include <StdIO.h>  
#include <Video.h>  
#include <TextUtils.h>  
#include <Strings.h>  
#include <DriverServices.h>  
#endif  
#endif  
  
#include <stdlib.h>  
#include "my_macros.h"  
#include "my_strings.h"  
  
// requestFlags bit values in VideoRequestRec (example use: 1<<kAbsoluteRequestBit)  
enum  
{  
    kBitDepthPriorityBit          = 0,    // Bit depth setting has priority over resolution  
    kAbsoluteRequestBit          = 1,    // Available setting must match request  
    kShallowDepthBit             = 2,    // Match bit depth less than or equal to request  
    kMaximizeResBit              = 3,    // Match screen resolution greater than or equal to request  
    kAllValidModesBit            = 4,    // Match display with valid timing modes (may include modes which are not  
    marked as safe)  
};  
  
// availFlags bit values in VideoRequestRec (example use: 1<<kModeValidNotSafeBit)  
enum  
{  
    kModeValidNotSafeBit         = 0      // Available timing mode is valid but not safe (requires user confirmation  
    switch)  
};  
  
// video request structure  
struct VideoRequestRec  
{  
    GDHandle      screenDevice;           // <in/out> nil will force search of best device, otherwise search this dev  
only  
    short         reqBitDepth;            // <in>      requested bit depth  
    short         availBitDepth;          // <out>     available bit depth  
    unsigned long reqHorizontal;          // <in>      requested horizontal resolution  
    unsigned long reqVertical;            // <in>      requested vertical resolution  
    unsigned long availHorizontal;        // <out>     available horizontal resolution  
    unsigned long availVertical;          // <out>     available vertical resolution  
    unsigned long requestFlags;           // <in>      request flags  
    unsigned long availFlags;             // <out>     available mode flags  
    unsigned long displayMode;            // <out>     mode used to set the screen resolution  
    unsigned long depthMode;              // <out>     mode used to set the depth  
    VDSwitchInfoRec switchInfo;           // <out>     DM2.0 uses this rather than displayMode/depthMode combo  
};  
  
//-----  
//
```

```
// Internal defines, structs, typedefs, and routine declarations
//
//-----

struct DepthInfo
{
    VDSwitchInfoRec    depthSwitchInfo;    // This is the switch mode to choose this timing/depth
    VPBlock            depthVPBlock;        // VPBlock (including size, depth and format)
};

struct ListIteratorDataRec
{
    unsigned long      displayModeFlags;    //
    VDSwitchInfoRec    displayModeSwitchInfo; //
    VDResolutionInfoRec displayModeResolutionInfo; //
    VDTimingInfoRec    displayModeTimingInfo; // Contains timing flags and such
    unsigned long      depthBlockCount;     // How many depths available for a particular timing
    struct DepthInfo   *depthBlocks;        // Array of DepthInfo
    Str255             displayModeName;     // name of the timing mode
};

struct display_specs
{
    Point    resolution; // display manager
    int      depth;      // display manager
    int      frequency;  // display manager
};

#ifdef __cplusplus
extern "C" {
#endif

void      PrintCurrentVideoSetting( GDHandle );
void      DisplayVideoSettings (void);
void      PrintAvailableVideoSettingsDM1( GDHandle );
void      PrintAvailableVideoSettingsDM2( GDHandle, DMDisplayModeListIteratorUPP, DMLIndexType, DMLListType );
pascal void ModeListIterator( void *, DMLIndexType, DMDisplayModeListEntryPtr );

Point      do_Get_Video_Resolution_By_GDHandle( GDHandle );
Point      do_Get_Video_Resolution_By_DisplayID( DisplayIDType );
Point      do_Get_Main_Video_Resolution( void );

int         do_Get_Video_Depth_By_GDHandle( GDHandle );
int         do_Get_Video_Depth_By_DisplayID( DisplayIDType );
int         do_Get_Main_Video_Depth( void );

int         do_Get_Video_Frequency_By_GDHandle( GDHandle );
int         do_Get_Video_Frequency_By_DisplayID( DisplayIDType );
int         do_Get_Main_Video_Frequency( void );

OSErr      do_Get_Display_Specs( DisplayIDType, struct display_specs * );

void      do_Test_Display_APIS(void);

#ifdef __cplusplus
}
#endif

#endif /* __my_displays__ */
```

```
//  
//  
//  
#include "my_displays.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
//  
#if TARGET_API_MAC_CARBON  
  
void DisplayVideoSettings( void )  
{  
    OSStatus err = noErr;  
  
    // We start with the first screen device  
    GDHandle walkDevice = DMGetFirstScreenDevice( dmOnlyActiveDisplays );  
    DMDisplayModeListIteratorUPP myModeIteratorProc = NewDMDisplayModeListIteratorUPP( ModeListIterator );  
    if( myModeIteratorProc && walkDevice )  
    {  
        short count = 0;  
  
        // Note that we are hosed if somebody changes the gdevice list behind our backs while we are iterating....  
        // For each screen device...  
        do  
        {  
            count++; // We count each screen device we find  
  
            DisplayIDType theDisplayID;  
            err = DMGetDisplayIDByGDevice( walkDevice, &theDisplayID, false );  
  
            DEBUG_PRINT("-----");  
            DEBUG_VAR_PRINT("GDevice #&d", count );  
            DEBUG_EXTRA_VAR_PRINT(" at location (&d", (long)(*walkDevice)->gdRect.left );  
            DEBUG_EXTRA_VAR_PRINT(", &d)", (long)(*walkDevice)->gdRect.top );  
            DEBUG_VAR_PRINT("DisplayID: &d", theDisplayID);  
            DEBUG_EXTRA_VAR_PRINT(" GDevice: 0x&X", *walkDevice );  
            DEBUG_PRINT("-----");  
  
            if( noErr == err )  
            {  
                DMLIndexType theDisplayModeCount = 0;  
                DMLType theDisplayModeList;  
  
                // This returns a pointer to the complete list of display modes.  
                // The list is a bunch of DMDisplayModeListEntryRec structs, accessed with the  
                // accessor function DMGetIndexedDisplayModeFromList().  
                err = DMNewDisplayModeList( theDisplayID, 0, 0, &theDisplayModeCount, &theDisplayModeList );  
                if( noErr == err )  
                {  
                    DEBUG_VAR_PRINT("Device has &d resolution mode(s)", theDisplayModeCount);  
                    PrintCurrentVideoSetting( walkDevice );  
                    PrintAvailableVideoSettingsDM2( walkDevice, myModeIteratorProc, theDisplayModeCount,  
theDisplayModeList );  
                    DMDisposeList( theDisplayModeList );  
                }  
                else  
                {  
                    DEBUG_PRINT("Error getting the display mode list");  
                }  
            }  
  
            walkDevice = DMGetNextScreenDevice ( walkDevice, dmOnlyActiveDisplays );  
        } while ( nil != walkDevice ); // go until no more gdevices  
  
        DisposeDMDisplayModeListIteratorUPP( myModeIteratorProc );  
    }  
}  
  
#else  
  
void DisplayVideoSettings( void )  
{  
    Boolean displayMgrPresent;  
    short iCount = 0; // just a counter of GDevices we have seen  
    DMDisplayModeListIteratorUPP myModeIteratorProc = nil; // for DM2.0 searches  
    SpBlock;  
    Boolean suppliedGDevice;  
    DisplayIDType theDisplayID; // for DM2.0 searches  
    DMLIndexType theDisplayModeCount; // for DM2.0 searches  
    DMLType theDisplayModeList; // for DM2.0 searches  
    long value = 0;  
    GDHandle walkDevice = nil; // for everybody  
  
    Gestalt( gestaltDisplayMgrAttr, &value );  
    displayMgrPresent = value & ( 1 << gestaltDisplayMgrPresent );
```

```
displayMgrPresent = displayMgrPresent && ( SVersion&spBlock) == noErr ); // need slot manager and Display
Manager
// if( do_Check_For_Display_Manager( 0x0200 ) )
if( displayMgrPresent )
{
    walkDevice = DMGetFirstScreenDevice( dmOnlyActiveDisplays );
    suppliedGDevice = false;
    myModeIteratorProc = NewDMDisplayModeListIteratorUPP( ModeListIterator ); // for DM2.0 searches
    if( myModeIteratorProc )
    {
        // Note that we are hosed if somebody changes the gdevice list behind our backs while we are
        iterating...
        // ...now do the loop if we can start
        if( walkDevice )
        {
            do // start the search
            {
                iCount++; // GDevice we are looking at (just a counter)
                DEBUG_PRINT("-----");
                DEBUG_VAR_PRINT("GDevice #&d", iCount );
                DEBUG_EXTRA_VAR_PRINT(" (0x%X)", *walkDevice );
                DEBUG_EXTRA_VAR_PRINT(" at location (&d", (long)(*walkDevice)->gdRect.left );
                DEBUG_EXTRA_VAR_PRINT(",&d", (long)(*walkDevice)->gdRect.top );
                DEBUG_PRINT("-----");

                PrintCurrentVideoSetting( walkDevice );

                if( noErr == DMGetDisplayIDByGDevice( walkDevice, &theDisplayID, false ) ) // DM1.0 does not
                need this, but it fits in the loop
                {
                    theDisplayModeCount = 0; // for DM2.0 searches
                    if( noErr == DMNewDisplayModeList( theDisplayID, 0, 0, &theDisplayModeCount,
&theDisplayModeList ) )
                    {
                        // search NuBus & PCI the new kool way through Display Manager 2.0
                        DEBUG_PRINT("Available Video Settings DM2.0 way");
                        PrintAvailableVideoSettingsDM2( walkDevice, myModeIteratorProc, theDisplayModeCount,
theDisplayModeList );
                        DMDDisposeList( theDisplayModeList ); // now toss the lists for this gdevice and go o
the next one
                    }
                    else
                    {
                        // search NuBus only the old disgusting way through the slot manager
                        DEBUG_PRINT("Available Video Settings DM1.0 way");
                        PrintAvailableVideoSettingsDM1( walkDevice );
                    }
                }
            } while ( !suppliedGDevice && nil != ( walkDevice = DMGetNextScreenDevice ( walkDevice,
dmOnlyActiveDisplays ) ) ); // go until no more gdevices
        }
        DisposedDMDisplayModeListIteratorUPP( myModeIteratorProc );
    }
}

#endif

//
pascal void ModeListIterator( void *userData, DMListIndexType, DMDisplayModeListEntryPtr displaymodeInfo )
{
    struct ListIteratorDataRec *userDataPtr = (struct ListIteratorDataRec*) userData;

    // This iterator function gives us a pointer to a struct DMDisplayModeListEntryRec.
    userDataPtr->displayModeFlags = displaymodeInfo->displayModeFlags; // Info on this particu
display mode
    userDataPtr->displayModeSwitchInfo = *displaymodeInfo->displayModeSwitchInfo; // not needed - depth
info has this per depth
    userDataPtr->displayModeResolutionInfo = *displaymodeInfo->displayModeResolutionInfo; // refresh rate,
pixels/lines at max depth
    userDataPtr->displayModeTimingInfo = *displaymodeInfo->displayModeTimingInfo; // to get the flags on
timing mode
    do_p_strcpy( (unsigned char *)&(userDataPtr->displayModeName), (const unsigned char
*)displaymodeInfo->displayModeName ); // the name of the mode

    // Get the DMDepthInfo into memory we own
    userDataPtr->depthBlockCount = displaymodeInfo->displayModeDepthBlockInfo->depthBlockCount;
    userDataPtr->depthBlocks = (struct DepthInfo *)NewPtrClear(userDataPtr->depthBlockCount * sizeof(struct
DepthInfo));
    if( ( NULL != userDataPtr->depthBlocks ) && userDataPtr->depthBlockCount )
    {
        for ( short count=0; count < userDataPtr->depthBlockCount; count++ )
        {
            userDataPtr->depthBlocks[count].depthSwitchInfo =
*displaymodeInfo->displayModeDepthBlockInfo->depthVPBlock[count].depthSwitchInfo;
```

```
        userDataPtr->depthBlocks[count].depthVPBlock =
*displayModeInfo->displayModeDepthBlockInfo->depthVPBlock[count].depthVPBlock;
    }
}

//
// Since Display Manager 2.0 appeared with System 7.5 Upgrade 2.0,
// it's definitely present under Carbon. On pre-Carbon systems,
// we check for the current version and do the appropriate thing.

#if TARGET_API_MAC_CARBON

void PrintCurrentVideoSetting( GDHandle walkDevice )
{
    VDSwitchInfoRec    switchInfo;

    OSErr err = DMGetDisplayMode( walkDevice, &switchInfo );
    if( noErr == err )
    {
        DEBUG_PRINT("Current settings:");
        DEBUG_VAR_PRINT("    Timing mode (csData): %u", switchInfo.csData );
        DEBUG_VAR_PRINT("    Depth mode (csMode): %u", switchInfo.csMode );
    }
}

#else

void PrintCurrentVideoSetting( GDHandle walkDevice )
{
    unsigned long      displayMgrVersion;
    OSErr              error = paramErr;
    CntrlParam         pBlock;
    VDSwitchInfoRec    switchInfo;
    AuxDCEHandle       theDCE;
    VDSwitchInfoRec    videoMode;

    Gestalt( gestaltDisplayMgrVers, (long*)&displayMgrVersion );
    if( displayMgrVersion >= 0x00020000 )
    {
        // Get the info the DM 2.0 way
        error = DMGetDisplayMode(walkDevice, &switchInfo);
        if( noErr == error )
        {
            DEBUG_PRINT("Current settings the DM 2.0 way...");
            DEBUG_VAR_PRINT("    Timing mode (csData): %u", switchInfo.csData );
            DEBUG_VAR_PRINT("    Depth mode (csMode): %u", switchInfo.csMode );
        }
    }
    else
    {
        // Get the info the DM 1.0 way
        videoMode.csMode = -1;          // init to bogus value
        videoMode.csData = -1;         // init to bogus value
        pBlock.ioNamePtr = nil;
        pBlock.ioCRefNum = (*(walkDevice))->gdRefNum;
        pBlock.csCode = cscGetCurMode;
        *(Ptr *)&pBlock.csParam[0] = (Ptr)&videoMode;

        // Ask the driver first since we trust it the most
        error = PBStatusSync((ParmBlkPtr)&pBlock);
        if( noErr == error && ( ( -1 == videoMode.csMode ) || ( -1 == videoMode.csData ) ) )
            error = statusErr;

        // If the driver has no clue, fill videoMode by hand as a last resort
        if( noErr != error )
        {
            theDCE = (AuxDCEHandle)GetDctlEntry((*(walkDevice))->gdRefNum);
            if( theDCE )
            {
                videoMode.csData = (unsigned char)(*(theDCE)->dCtlSlotId;
                videoMode.csMode = (*(walkDevice))->gdMode;
                error = noErr;
            }
        }

        if( noErr == error )
        {
            DEBUG_PRINT("Current settings the DM 1.0 way...");
            DEBUG_VAR_PRINT("    Timing mode (csData): %u", videoMode.csData );
            DEBUG_VAR_PRINT("    Depth mode (csMode): %u", videoMode.csMode );
        }
    }
}

#endif

//
```

```
void PrintAvailableVideoSettingsDM2( GDHandle walkDevice, DMDisplayModeListIteratorUPP myModeIteratorProc,
                                     DMListIndexType theDisplayModeCount, DMListType theDisplayModeList )
{
    struct ListIteratorDataRec      searchData;
    double_t                       refreshRate;
    OSErr                          err = noErr;

    searchData.depthBlocks = nil;
    // First, we go through each video mode. A video mode is a unique combination of resolution and refresh rate.
    for( short count=0; count < theDisplayModeCount; count++ )
    {
        // Each mode may have a number of bit depths.
        // The searchData is a subset of a struct DMDisplayModeListEntryRec.
        DMGetIndexedDisplayModeFromList( theDisplayModeList, count, NULL, myModeIteratorProc, &searchData );

        DEBUG_PRINT("-----");
        DEBUG_EXTRA_VAR_PRINT("Timing mode (csData): %u", searchData.displayModeSwitchInfo.csData ); // the same as
searchData.depthBlocks[0].depthSwitchInfo.csData
        DEBUG_EXTRA_VAR_PRINT(" (0x%X)", searchData.displayModeSwitchInfo.csData );
        DEBUG_EXTRA_VAR_PRINT(" named \"%s\"", do_p2c_str(searchData.displayModeName) );
        DEBUG_EXTRA_PRINT("-----");

        // DEBUG_VAR_PRINT("displayModeSwitchInfo (%#010X)", searchData.displayModeSwitchInfo );
        // DEBUG_VAR_PRINT("csMode: %d", searchData.depthBlocks[0].depthSwitchInfo.csMode );

        refreshRate = Fix2X( searchData.displayModeResolutionInfo.csRefreshRate );
        refreshRate = round( refreshRate );
        if( refreshRate == 0 ) {
            DEBUG_PRINT("Refresh rate: 0 (not defined in displayModeResolutionInfo.csRefreshRate)");
        } else {
            DEBUG_VAR_PRINT("Refresh rate: %g", refreshRate );
        }

        if( searchData.displayModeResolutionInfo.csResolutionFlags & 1 << kResolutionHasMultipleDepthSizes ) {
            DEBUG_PRINT("DisplayMode has different H&V per bit depth");
        } else {
            DEBUG_PRINT("DisplayMode does not have different H&V per bit depth");
        }

        {
            char          tempArr[6];
            ResType*      tempPtr = (ResType*) &tempArr[0];           // Make a convenient ptr to assign the
            *tempPtr = searchData.displayModeTimingInfo.csTimingFormat; // contents of string are the resType
            tempArr[4] = 0;                                           // null temp the string
            DEBUG_VAR_PRINT("Timing format: \"%s\"", tempArr );
            DEBUG_VAR_PRINT("Timing csData %d", searchData.displayModeTimingInfo.csTimingData );
        }

        // These seem to be the same for aLL the depth modes
        Boolean modeOk = false;
        unsigned long switchFlags;
        err = DMCheckDisplayMode( walkDevice,
                                   searchData.displayModeSwitchInfo.csData,
                                   searchData.displayModeSwitchInfo.csMode,
                                   &switchFlags,
                                   0,
                                   &modeOk);

        // switch flags
        if( noErr == err && modeOk )
        {
            DEBUG_VAR_PRINT("Switch flags (0x%X):", switchFlags);

            if( switchFlags & ( 1 << kNoSwitchConfirmBit ) ) {
                DEBUG_EXTRA_PRINT(" Confirmation not required");
            } else {
                DEBUG_EXTRA_PRINT(" Confirmation required");
            }

            if( switchFlags & ( 1 << kDepthNotAvailableBit ) ) {
                DEBUG_EXTRA_PRINT(", Current depth not available in this mode");
            } else {
                DEBUG_EXTRA_PRINT(", Current depth available in this mode");
            }

            if( switchFlags & ( 1 << kShowModeBit ) ) {
                DEBUG_EXTRA_PRINT(", Always shown");
            } else {
                DEBUG_EXTRA_PRINT(", Not always shown");
            }

            if( switchFlags & ( 1 << kModeNotResizeBit ) ) {
                DEBUG_EXTRA_PRINT(", Not resizeable");
            } else {
                DEBUG_EXTRA_PRINT(", Resizeable");
            }
        }
    }
}
```



```
// timing flags
{
    DEBUG_VAR_PRINT("Timing flags (0x%X):", searchData.displayModeTimingInfo.csTimingFlags);

    if(searchData.displayModeTimingInfo.csTimingFlags & 1<<kModeValid) {
        DEBUG_EXTRA_PRINT(" Valid");
    } else {
        DEBUG_EXTRA_PRINT(" Invalid");
    }

    if(searchData.displayModeTimingInfo.csTimingFlags & 1<<kModeSafe) {
        DEBUG_EXTRA_PRINT(", Safe");
    } else {
        DEBUG_EXTRA_PRINT(", Unsafe");
    }

    if(searchData.displayModeTimingInfo.csTimingFlags & 1<<kModeDefault) {
        DEBUG_EXTRA_PRINT(", Default");
    } else {
        DEBUG_EXTRA_PRINT(", Not default");
    }

    if(searchData.displayModeTimingInfo.csTimingFlags & 1<<kModeShowNow) {
        DEBUG_EXTRA_PRINT(", Always shown");
    } else {
        DEBUG_EXTRA_PRINT(", Not always shown");
    }

    if(searchData.displayModeTimingInfo.csTimingFlags & 1<<kModeNotResize) {
        DEBUG_EXTRA_PRINT(", Not resizable");
    } else {
        DEBUG_EXTRA_PRINT(", Resizable");
    }

    if(searchData.displayModeTimingInfo.csTimingFlags & 1<<kModeRequiresPan) {
        DEBUG_EXTRA_PRINT(", Requires pan");
    } else {
        DEBUG_EXTRA_PRINT(", No pan");
    }
}

// mode flags
{
    DEBUG_VAR_PRINT("Mode flags (0x%X):", searchData.displayModeFlags);

    if( searchData.displayModeFlags & ( 1 << 0 ) ) {
        DEBUG_EXTRA_PRINT(" Stripped");
    } else {
        DEBUG_EXTRA_PRINT(" Not Stripped");
    }
}

DEBUG_PRINT("Available depths:");

if( searchData.depthBlockCount )
{
    // We print out info for each bit depth and refresh rate
    for ( short kCount = 0; kCount < searchData.depthBlockCount; kCount++ )
    {
        // print all the timing information
        DEBUG_VAR_PRINT(" Depth: %d",
searchData.depthBlocks[kCount].depthVPBlock.vpPixelSize );
        DEBUG_EXTRA_VAR_PRINT(" Depth mode (csMode): %d",
searchData.depthBlocks[kCount].depthSwitchInfo.csMode );
        DEBUG_EXTRA_VAR_PRINT(" (0x%X)",
searchData.depthBlocks[kCount].depthSwitchInfo.
);
        DEBUG_EXTRA_VAR_PRINT(" Resolution: %dh",
searchData.depthBlocks[kCount].depthVPBlock.vpBounds.right );
        DEBUG_EXTRA_VAR_PRINT(" x %dv",
searchData.depthBlocks[kCount].depthVPBlock.vpBounds.bottom );
        DEBUG_EXTRA_VAR_PRINT(" Components per pixel: %d",
searchData.depthBlocks[kCount].depthVPBlock.vpCmpCount );
        DEBUG_EXTRA_VAR_PRINT(" Bits per component: %d",
searchData.depthBlocks[kCount].depthVPBlock.vpCmpSize );
        // DEBUG_VAR_PRINT(" depthSwitchInfo (%#010X)",
searchData.depthBlocks[kCount].depthSwitchInfo
    )
    }

    if( searchData.depthBlocks )
    {
        DisposePtr ((Ptr)searchData.depthBlocks); // Allocated in the mode list iterator; disposed of here.
        searchData.depthBlocks = NULL;
    }
}

if( theDisplayModeCount > 0 )
    DEBUG_PRINT("----- "); // draw line at the end
```

```
}  
//  
#if !TARGET_API_MAC_CARBON  
  
void PrintAvailableVideoSettingsDM1( GDHandle walkDevice )  
{  
    AuxDCEHandle    myAuxDCEHandle;  
    unsigned long    depthMode;  
    unsigned long    displayMode;  
    OSErr            error;  
    OSErr            errorEndOfTimings;  
    short            height;  
    Boolean          modeOk;  
    SpBlock          spAuxBlock;  
    SpBlock          spBlock;  
    unsigned long    switchFlags;  
    VPBlock          *vpData;  
    short            width;  
  
    myAuxDCEHandle = (AuxDCEHandle) GetDctlEntry((**walkDevice).gdRefNum);  
    spBlock.spSlot = (**myAuxDCEHandle).dctlSlot;  
    spBlock.spID = (**myAuxDCEHandle).dctlSlotId;  
    spBlock.spExtDev = (**myAuxDCEHandle).dctlExtDev;  
    spBlock.spHwDev = 0; // we are going to get this pup  
    spBlock.spParamData = 1<<foneslot; // this slot, enabled, and it better be here.  
    spBlock.spTBMask = 3; // don't have constants for this yet  
    errorEndOfTimings = SGetTypeSRSrc(&spBlock); // get the spDrvrHW so we know the ID of this puppy. This is  
    // important // since some video cards support more than one display, an  
    // spDrvrHW // ID can, and will, be used to differentiate them.  
  
    if( noErr == errorEndOfTimings )  
    {  
        // reinit the param block for the SGetTypeSRSrc loop, keep the spDrvrHW we just got  
        spBlock.spID = 0; // start at zero,  
        spBlock.spTBMask = 2; // 0b0010 - ignore DrvrSW - why ignore the SW side? Is it n  
        // important for video? // so we even get 640x399 on Blackbird  
        spBlock.spParamData = (1<<fall) + (1<<foneslot) + (1<<fnext); // 0b0111 - this slot, enabled or disabled,  
        spBlock.spCategory=catDisplay;  
        spBlock.spCType=typeVideo;  
        errorEndOfTimings = SGetTypeSRSrc(&spBlock); // but only on 7.0 systems, not a problem since we require  
        DM1.0  
  
        // now, loop through all the timings for this GDevice  
        if( noErr == errorEndOfTimings ) do  
        {  
            // now, loop through all possible depth modes for this timing mode  
            displayMode = (unsigned char)spBlock.spID; // "timing mode, ie:resource ref number"  
            for( short jCount = firstVidMode; jCount<= sixthVidMode; jCount++ )  
            {  
                depthMode = jCount; // vid mode  
                error = DMCheckDisplayMode(walkDevice,displayMode,depthMode,&switchFlags,0,&modeOk);  
  
                // only if the mode okay  
                if(noErr == error && modeOk)  
                {  
                    // have a good displayMode/depthMode combo - now lets look inside  
                    spAuxBlock = spBlock; // don't ruin the iteration spBlock!!  
                    spAuxBlock.spID = depthMode; // vid mode  
                    error=SFindStruct(&spAuxBlock); // get back a new spsPointer  
                    if(noErr == error) // keep going if no error..  
                    {  
                        spAuxBlock.spID = 0x01; // mVidParams request  
                        error=SGetBlock (&spAuxBlock); // use the new spPointer and get back...a NewPtr'ed spResul  
                        if(noErr == error) // ..keep going if no error..  
                        {  
                            // We have data! lets have a look  
                            vpData = (VPBlock*)spAuxBlock.spResult;  
                            height = vpData->vpBounds.bottom; // left and top are usually zero  
                            width = vpData->vpBounds.right;  
  
                            // print screen data  
                            DEBUG_VAR_PRINT("Timing Mode: %d", displayMode );  
                            DEBUG_VAR_PRINT("Depth Mode: %d", depthMode );  
                            DEBUG_VAR_PRINT("Depth: %d", vpData->vpPixelSize );  
                            DEBUG_VAR_PRINT("Resolution: %dh", vpData->vpBounds.right );  
                            DEBUG_EXTRA_VAR_PRINT(" x %dv", vpData->vpBounds.bottom );  
                            DEBUG_VAR_PRINT("Components per Pixel: %d", vpData->vpCmpCount );  
                            DEBUG_VAR_PRINT("Bits per Component: %d", vpData->vpCmpSize );  
                            DEBUG_PRINT("Switch flags:");  
  
                            if( switchFlags & ( 1 << kNoSwitchConfirmBit ) ) {  
                                DEBUG_PRINT(" Confirmation not required");  
                            } else {  
                                DEBUG_PRINT(" Confirmation required");  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```
        if( switchFlags & ( 1 << kDepthNotAvailableBit ) ) {
            DEBUG_PRINT(" Current depth not available in this mode");
        } else {
            DEBUG_PRINT(" Current depth available in this mode");
        }

        if( switchFlags & ( 1 << kShowModeBit ) ) {
            DEBUG_PRINT(" Always shown");
        } else {
            DEBUG_PRINT(" Not always shown");
        }

        if( switchFlags & ( 1 << kModeNotResizeBit ) ) {
            DEBUG_PRINT(" Not resizeable");
        } else {
            DEBUG_PRINT(" Resizeable");
        }

        if( spAuxBlock.spResult )
            DisposePtr( (Ptr)spAuxBlock.spResult ); // toss this puppy when done
    }
}

// go around again, looking for timing modes for this GDevice
spBlock.spTBMask = 2; // ignore DrvrSW
spBlock.spParamData = (1<<fall) + (1<<foneslot) + (1<<fnext); // next resource, this slot, whether
enabled or disabled
errorEndOfTimings = SGetTypeSRsrc(&spBlock); // and get the next timing mode
} while ( noErr == errorEndOfTimings ); // until the end of this GDevice
}
}

#endif

//
#pragma mark -
//

Point do_Get_Video_Resolution_By_GDHandle( GDHandle device_handle )
{
    Point res = {0,0};

    if( NULL != device_handle )
    {
        res.h = (*device_handle)->gdRect.right - (*device_handle)->gdRect.left;
        res.v = (*device_handle)->gdRect.bottom - (*device_handle)->gdRect.top;
        DEBUG_VAR_PRINT("Video resolution: %dh",res.h);
        DEBUG_EXTRA_VAR_PRINT(" x %dv",res.v);
    }

    return( res );
}

//
Point do_Get_Video_Resolution_By_DisplayID( DisplayIDType display_id )
{
    Point res = {0,0};

    GDHandle device_handle = NULL;
    OSStatus err = DMGetGDeviceByDisplayID( display_id, &device_handle, false );
    if( noErr == err && NULL != device_handle )
    {
        res = do_Get_Video_Resolution_By_GDHandle( device_handle );
    }

    return( res );
}

//
Point do_Get_Main_Video_Resolution( void )
{
    return( do_Get_Video_Resolution_By_GDHandle( GetMainDevice() ) );
}

//
#pragma mark -
//

int do_Get_Video_Depth_By_GDHandle( GDHandle device_handle )
{
    if( NULL == device_handle )
        return paramErr;
}
```

```
DisplayIDType display_id;
OSErr err = DMGetDisplayIDByGDevice( device_handle, &display_id, false );
if( err )
    return err;

return( do_Get_Video_Depth_By_DisplayID( display_id ) );
}

// _____

int do_Get_Video_Depth_By_DisplayID( DisplayIDType display_id )
{
    struct display_specs the_specs;
    the_specs.depth = 0;

    OSErr err = do_Get_Display_Specs( display_id, &the_specs );
    if( err )
        return( err );

    DEBUG_VAR_PRINT("Video depth: %d bit",the_specs.depth);
    return( the_specs.depth );
}

// _____

int do_Get_Main_Video_Depth( void )
{
    return( do_Get_Video_Depth_By_GDHandle( GetMainDevice() ) );
}

// _____
#pragma mark -
// _____

int do_Get_Video_Frequency_By_GDHandle( GDHandle device_handle )
{
    if( NULL == device_handle )
        return( paramErr );

    DisplayIDType display_id;
    OSErr err = DMGetDisplayIDByGDevice( device_handle, &display_id, false );
    if( err )
        return( err );

    return( do_Get_Video_Frequency_By_DisplayID( display_id ) );
}

// _____

int do_Get_Video_Frequency_By_DisplayID( DisplayIDType display_id )
{
    struct display_specs the_specs;
    the_specs.frequency = 0;

    OSErr err = do_Get_Display_Specs( display_id, &the_specs );
    if( err )
        return( err );

    DEBUG_VAR_PRINT("Video frequency: %d Hz",the_specs.frequency);
    return( the_specs.frequency );
}

// _____

int do_Get_Main_Video_Frequency( void )
{
    return( do_Get_Video_Frequency_By_GDHandle( GetMainDevice() ) );
}

// _____
#pragma mark -
// _____

OSErr do_Get_Display_Specs( DisplayIDType display_id, struct display_specs *the_specs )
{
    OSErr err = noErr;

    if( NULL == the_specs )
        return paramErr;

    GDHandle device_handle = NULL;
    err = DMGetGDeviceByDisplayID( display_id, &device_handle, false );
    if( noErr != err && NULL == device_handle )
        return err;

    VDSwitchInfoRec switch_info;
```

```
err = DMGetDisplayMode( device_handle, &switch_info );
if( err )
    return err;

DEBUG_VAR_PRINT("DisplayID: %d",display_id);
DEBUG_EXTRA_VAR_PRINT(" GDevice: 0x%X", *device_handle );
DEBUG_VAR_PRINT(" Timing mode (csData): %u", switch_info.csData );
DEBUG_VAR_PRINT(" Depth mode (csMode): %u", switch_info.csMode );

DMDisplayModeListIteratorUPP myModeIteratorProc = NewDMDisplayModeListIteratorUPP( ModeListIterator );
if( myModeIteratorProc )
{
    DMListIndexType theDisplayModeCount = 0;
    DMListType theDisplayModeList = NULL;
    err = DMNewDisplayModeList( display_id, 0, 0, &theDisplayModeCount, &theDisplayModeList );
    if( noErr == err && NULL != theDisplayModeList )
    {
        Boolean found_it = false;

        // For each video mode in the list...
        for( short i=0; i < theDisplayModeCount; i++ )
        {
            struct ListIteratorDataRec searchData;
            searchData.depthBlocks = NULL;
            // We pull one out and look at all of its different depths.
            err = DMGetIndexedDisplayModeFromList( theDisplayModeList, i, NULL, myModeIteratorProc, &searchData
);
            if( noErr == err && searchData.depthBlockCount && searchData.depthBlocks )
            {
                // Unfortunately, we have to look through all depths of a mode to find the one that matches the
current mode.
                for ( short j = 0; j < searchData.depthBlockCount; j++ )
                {
                    DEBUG_VAR_PRINT(" Timing mode (csData): %u", searchData.depthBlocks[j].depthSwitchInfo.csD
);
                    DEBUG_VAR_PRINT(" Depth mode (csMode): %u", searchData.depthBlocks[j].depthSwitchInfo.csMo
);

                    // The current "timing mode" is the one we're after. Due to a change in the timing
// format of some video drivers, we have to test the format being used and access
// the data we're after differently.

                    if( switch_info.csMode < 128 ) // this means we got a bit depth in the csMode instead of a

// depth mode

                    UInt32 timing_mode = 0;
                    if( kDetailedTimingFormat == searchData.displayModeTimingInfo.csTimingFormat )
                        timing_mode = switch_info.csData; // use detailed timing modes
                    else // kDeclROMtables
                        timing_mode = switch_info.csMode; // uses timing mode plus old stye depth modes

                    if( searchData.depthBlocks[j].depthSwitchInfo.csMode == timing_mode )
                    {
                        the_specs->resolution.h = (*device_handle)->gdRect.right - (*device_handle)->gdRect
                        the_specs->resolution.v = (*device_handle)->gdRect.bottom - (*device_handle)->gdRect
                        the_specs->depth = searchData.depthBlocks[j].depthVPBlock.vpPixelSize;
                        the_specs->frequency = FixedToInt( searchData.displayModeResolutionInfo.csRefres

);

                        DEBUG_VAR_PRINT(" Resolution: %dh",the_specs->resolution.h);
                        DEBUG_EXTRA_VAR_PRINT(" x %dv",the_specs->resolution.v);
                        DEBUG_VAR_PRINT(" Bit depth: %d bit", the_specs->depth );
                        DEBUG_VAR_PRINT(" Refresh rate: %d", the_specs->frequency );

                        found_it = true;
                        break;
                    }
                }

                DisposePtr ((Ptr)searchData.depthBlocks); // Allocated in the mode list iterator; disposed of
searchData.depthBlocks = NULL;
            }

            if( found_it )
                break;
        }

        DMDisposeList(theDisplayModeList);
    }

    DisposeDMDisplayModeListIteratorUPP( myModeIteratorProc );
}

return( err );
}

//
#pragma mark -
```

// _____

```
void do_Test_Display_APIS()  
{  
    DEBUG_PRINT("----- Testing my_displays.c APIs -----");  
  
    DisplayVideoSettings();  
  
    GDHandle device_handle = GetMainDevice();  
    DisplayIDType display_id;  
    DMGetDisplayIDByGDevice( device_handle, &display_id, false );  
  
    DEBUG_VAR_PRINT("Main Device DisplayID: %d",display_id);  
    DEBUG_EXTRA_VAR_PRINT(" GDevice: 0x%X",*device_handle);  
  
    DEBUG_PRINT("----- do_Get_Main_Video_Depth()");  
    do_Get_Main_Video_Depth();  
  
    DEBUG_PRINT("----- do_Get_Video_Depth_By_GDHandle()");  
    do_Get_Video_Depth_By_GDHandle( device_handle );  
  
    DEBUG_PRINT("----- do_Get_Video_Depth_By_DisplayID()");  
    do_Get_Video_Depth_By_DisplayID( display_id );  
  
    DEBUG_PRINT("----- do_Get_Main_Video_Frequency()");  
    do_Get_Main_Video_Frequency();  
  
    DEBUG_PRINT("----- do_Get_Video_Frequency_By_GDHandle()");  
    do_Get_Video_Frequency_By_GDHandle( device_handle );  
  
    DEBUG_PRINT("----- do_Get_Video_Frequency_By_DisplayID()");  
    do_Get_Video_Frequency_By_DisplayID( display_id );  
  
    DEBUG_PRINT("----- do_Get_Main_Video_Resolution()");  
    do_Get_Main_Video_Resolution();  
  
    DEBUG_PRINT("----- do_Get_Video_Resolution_By_GDHandle()");  
    do_Get_Video_Resolution_By_GDHandle( device_handle );  
  
    DEBUG_PRINT("----- do_Get_Video_Resolution_By_DisplayID()");  
    do_Get_Video_Resolution_By_DisplayID( display_id );  
  
    DEBUG_PRINT("----- Finished Testing APIs -----");  
}
```

```
//  
// _____  
// _____ ©1998 bergdesign inc.  
// _____  
  
#ifndef __my_files__  
#define __my_files__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
  
#include <ctype.h>  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Files.h>  
#include <LowMem.h>  
#include <Navigation.h>  
#include <AppleEvents.h>  
#endif  
#endif  
  
#include "MoreFilesExtras.h"  
#include "my_dialogs.h"  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
#define LAST_CHAR(s) (s)[strlen(s)-1]  
  
enum  
{  
    kMaxNumFileNameCharacters = 31  
};  
  
enum  
{  
    Extension_ = 1,  
    Filename_ = 2,  
    Directory_ = 4,  
    Drive_ = 8,  
    Wildname_ = 16,  
    Wildpath_ = 32  
};  
  
#define rOpenResource 128  
#define dontSaveChanges 3  
  
OSErr do_Nav_Choose_Folder ( const unsigned char *, FSSpec * );  
OSErr do_Nav_Choose_Object ( const unsigned char *, FSSpec * );  
  
OSStatus do_Nav_Get_File(OSType, short, OSType *, UInt32, NavEventProcPtr, FSSpec *, OSType *);  
OSStatus do_Nav_Put_File(const unsigned char *, const unsigned char *, OSType, OSType, NavEventProcPtr, FSSpec *  
NavReplyRecord *);  
  
OSStatus do_Nav_Complete_Save(const FSSpec *, NavReplyRecord *);  
short do_Nav_Confirm_Save(const unsigned char *, Boolean, NavEventProcPtr);  
  
static NavTypeListHandle NewOpenHandle(OSType, short, OSType * );  
static OSStatus SendOpenAE(AEDescList list);  
OSErr do_AEGetDescData ( const AEDesc *, DescType *, void *, ByteCount, ByteCount * );  
  
// Callback to handle events that occur while navigation  
// dialogs are up but really should be handled by the application  
extern void HandleEvent(EventRecord * pEvent);  
pascal void MyEventProc(const NavEventCallbackMessage callBackSelector, NavCBRecPtr callBackParms,  
NavCallBackUserData callBackUD);  
  
OSErr do_Add_File_ID ( FSSpec *, long * );  
OSErr do_Get_File_ID ( FSSpec *, long * );  
  
OSErr do_Get_Maskerade_Sibling_FSSpec ( FSSpec *, FSSpec *, Boolean, Boolean, int );  
OSErr do_Get_App_FSSpec ( FSSpec * );  
  
OSErr do_Assert_Sibling_File_FSSpec ( FSSpec *, unsigned char *, FSSpec *, Boolean * );  
OSErr do_Assert_Child_File_FSSpec ( FSSpec *, unsigned char *, FSSpec *, Boolean * );  
  
OSErr do_Get_Parent_Folder ( FSSpec *, FSSpec * );  
OSErr do_Assert_Sibling_Folder ( FSSpec *, unsigned char *, FSSpec *, Boolean * );
```

```
Boolean    do_Filename_Has_Wildcard_Chars ( char * );

int         do_Split_DOS_Filename ( char *spec, char *drive, char *pname, char *path, char *fname, char *name, char
*ext);
char        *do_Merge_DOS_Filename ( char *spec, char *drive, char *pname, char *path, char *fname, char *name, cha
*ext);

char        *do_UNIX_To_DOS_Path ( char *path );
char        *do_DOS_To_UNIX_Path ( char *path );

char        *do_MAC_To_UNIX_Path ( char *path );
char        *do_UNIX_To_MAC_Path ( char *path );

#ifdef __cplusplus
}
#endif

#endif /* __my_files__ */
```



```
//  
// _____  
// _____ ©1998-2001 bergdesign inc.  
// _____  
  
#include "my_files.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
//extern Boolean gInModalState;  
  
// _____  
  
OSErr do_Nav_Choose_Folder ( const unsigned char *prompt, FSSpec *folder_fsspec )  
{  
    NavReplyRecord    the_reply;  
    NavDialogOptions  dialog_options;  
    OSErr             err = noErr;  
    NavEventUPP       nav_event_upp = NULL;  
  
    err = NavGetDefaultDialogOptions ( &dialog_options );  
    if ( err == noErr )  
    {  
        do_p_strcpy ( dialog_options.message, prompt);  
        dialog_options.preferenceKey = 0;  
  
        // nav_event_upp = NewNavEventProc ( myEventProc );  
  
        err = NavChooseFolder ( NULL, &the_reply, &dialog_options, nav_event_upp, NULL, NULL );  
  
        if ( nav_event_upp != NULL )  
        {  
            // DisposeRoutineDescriptor ( nav_event_upp );  
            DisposeNavEventUPP( nav_event_upp );  
        }  
  
        if ( ( err == noErr ) && ( the_reply.validRecord ) )  
        {  
            AEDesc result_desc;  
  
            // grab the target FSSpec from the AEDesc:  
            err = AECOerceDesc ( &(the_reply.selection), typeFSS, &result_desc );  
            if ( err == noErr )  
            {  
                DescType      desc_type;  
                ByteCount      actual_size = 0;  
  
                err = do_AEGetDescData ( &result_desc, &desc_type, folder_fsspec, sizeof ( FSSpec ), &actual_size )  
                {  
                    if ( err != noErr )  
                    {  
                        DEBUG_PRINT("Couldn't get the FSSpec of the chosen folder");  
                    }  
                }  
                AEDisposeDesc ( &result_desc );  
  
                err = NavDisposeReply ( &the_reply );  
            }  
        }  
  
        return ( err );  
    }  
}  
  
// _____  
  
OSErr do_Nav_Choose_Object ( const unsigned char *prompt, FSSpec *folder_fsspec )  
{  
    NavReplyRecord    the_reply;  
    NavDialogOptions  dialog_options;  
    OSErr             err = noErr;  
    NavEventUPP       nav_event_upp = NULL;  
  
    err = NavGetDefaultDialogOptions ( &dialog_options );  
    if ( err == noErr )  
    {  
        do_p_strcpy ( dialog_options.message, prompt);  
        dialog_options.preferenceKey = 0;  
  
        // nav_event_upp = NewNavEventProc ( myEventProc );  
  
        err = NavChooseObject ( NULL, &the_reply, &dialog_options, nav_event_upp, NULL, NULL );  
  
        if ( nav_event_upp != NULL )  
        {  
            // DisposeRoutineDescriptor ( nav_event_upp );  
            DisposeNavEventUPP( nav_event_upp );  
        }  
    }  
}
```

```
if ( ( err == noErr ) && ( the_reply.validRecord ) )
{
    AEDesc result_desc;

    // grab the target FSSpec from the AEDesc:
    err = AECoerceDesc ( &(the_reply.selection), typeFSS, &result_desc );
    if ( err == noErr )
    {
        DescType      desc_type;
        ByteCount      actual_size = 0;

        err = do_AEGetDescData ( &result_desc, &desc_type, folder_fsspec, sizeof ( FSSpec ), &actual_si
    );
        if ( err != noErr )
        {
            DEBUG_PRINT("Couldn't get the FSSpec of the chosen object");
        }
        AEDisposeDesc ( &result_desc );

        err = NavDisposeReply ( &the_reply );
    }
}

return ( err );
}

//
// Displays the NavGet dialog and returns the selected file location.
// To enable multiple document opening through AppleEvents pass NULL as the fileSpec and fileType.

OSStatus do_Nav_Get_File(    OStype app_signature,
                             short num_types,
                             OStype type_list[],
                             UInt32 option_flags,
                             NavEventProcPtr user_event_proc,
                             FSSpec* file_fsspec,
                             OStype* file_type)
{
    NavReplyRecord      nav_reply;
    NavDialogOptions    dialog_options;
    OSStatus            err = noErr;
    NavTypeListHandle   nav_open_list = NULL;
    NavEventUPP         nav_event_upp = NULL;

    err = NavGetDefaultDialogOptions(&dialog_options);
    if ( err == noErr )
    {
        // Identify the app in the dialog box window title
        BlockMoveData(LMGetCurAppName(), dialog_options.clientName, LMGetCurAppName()[0] + 1);

        if( option_flags != 0 )
        {
            dialog_options.dialogOptionFlags = option_flags;
        }
        else
        {
            // We could use the defaults, but they usually screw us up.
            // We override the defaults and set all our own values.
            dialog_options.dialogOptionFlags = 0;

            // These are all the Nav Services dialog options
            dialog_options.dialogOptionFlags |= kNavNoTypePopup;                // Don't show file type pop-up.
            dialog_options.dialogOptionFlags |= kNavDontAutoTranslate;          // Don't auto-translate on Open.
            dialog_options.dialogOptionFlags |= kNavDontAddTranslateItems;      // Don't add translation choices.

by default
//
// default
            dialog_options.dialogOptionFlags |= kNavAllFilesInPopup;            // Add "All Files" menu item.
            dialog_options.dialogOptionFlags |= kNavAllowStationery;            // Allow stationery files.

default
            dialog_options.dialogOptionFlags |= kNavAllowPreviews;              // Allow previews.

//
// by default
            dialog_options.dialogOptionFlags |= kNavAllowMultipleFiles;         // Allow multiple selection.

//
// selection.
            dialog_options.dialogOptionFlags |= kNavAllowInvisibleFiles;        // Show invisible objects.
            dialog_options.dialogOptionFlags |= kNavDontResolveAliases;          // Don't resolve aliases.
            dialog_options.dialogOptionFlags |= kNavSelectDefaultLocation;      // Make default location the browse

//
// selection.
            dialog_options.dialogOptionFlags |= kNavSelectAllReadableItem;      // Make All Readable Items default

//
// v2.0 or greater
            dialog_options.dialogOptionFlags |= kNavSupportPackages ;           // recognize file system packages,
            dialog_options.dialogOptionFlags |= kNavAllowOpenPackages;          // allow opening of packages, v2.0

greater
//
// recents list, v2.0 or greater
            dialog_options.dialogOptionFlags |= kNavDontAddRecents;             // don't add chosen objects to the
            dialog_options.dialogOptionFlags |= kNavDontUseCustomFrame;         // don't draw the custom area bevel
frame, v2.0 or greater
//
            dialog_options.dialogOptionFlags |= kNavDontConfirmReplacement;     // don't show the "Replace File?"
    }
}
```

```
alert on save conflict, v3.0 or greater
}

// Make it open in the same location as the last time it was opened
dialog_options.location.h = -1;
dialog_options.location.v = -1;

nav_reply.validRecord = false;

if ( ( NULL != type_list ) && ( num_types > 0 ) )
{
    // Turn the type list into a NavServices open list
    nav_open_list = NewOpenHandle( app_signature, num_types, type_list );
    if ( NULL != nav_open_list )
        HLock((Handle)nav_open_list);
}
else
{
    // Get the type list from the 'open' resource
    nav_open_list = (NavTypeListHandle)GetResource( 'open', rOpenResource );
}

if ( NULL != user_event_proc )
{
    nav_event_upp = NewNavEventUPP(user_event_proc);
}

// If nav_event_upp is NULL, then the dialog is not movable or resizable.
err = NavGetFile(NULL, &nav_reply, &dialog_options, nav_event_upp, NULL, NULL, nav_open_list, NULL);

if( NULL != nav_event_upp )
{
    DisposeNavEventUPP(nav_event_upp);
}

// If we used the type list, we dispose of the NavTypeListHandle
// differently than if we used an 'open' resource.
if ( NULL != nav_open_list )
{
    if ( ( NULL != type_list ) && ( num_types > 0 ) )
    {
        HUnlock((Handle)nav_open_list);
        DisposeHandle((Handle)nav_open_list);
    }
    else
    {
        ReleaseResource( (Handle)nav_open_list );
    }
}

if ( true == nav_reply.validRecord && noErr == err )
{
    // grab the target FSSpec from the AEDesc for opening:
    AEDesc resultDesc;
    FInfo fileInfo;

    // Is this a single file open
    if ( file_fsspec != NULL )
    {
        AEKeyword keyword;

        // We get the first item (which should be the only item).
        err = AEGetNthDesc( &nav_reply.selection, 1, typeFSS, &keyword, &resultDesc );
        if (err == noErr)
        {
            // And put it in the pointer passed into the function
            err = AEGetDescData( &resultDesc, file_fsspec, sizeof(FSSpec) );
        }

        // Now we get info for the file and determine its file type
        err = FSPGetFInfo( file_fsspec, &fileInfo );
        if (err == noErr)
        {
            if ( file_type != NULL )
                *file_type = fileInfo.fdType;
        }
    }
    else
    {
        // Multiple files open: use AppleEvents
        err = SendOpenAE(nav_reply.selection);
    }

    NavDisposeReply(&nav_reply);
}
else
{
    err = userCanceledErr;
}
```

```
    }  
}  
return err;  
}  
  
//  
// Displays the NavPut dialog and returns the selected file location and replacing info.  
  
OSStatus do_Nav_Put_File(const unsigned char *message, const unsigned char *fileName, OSType file_type, OSType  
fileCreator, NavEventProcPtr user_event_proc, FSSpec* file_fsspec, NavReplyRecord* reply)  
{  
    NavDialogOptions    dialog_options;  
    OSStatus            err = noErr;  
    NavEventUPP         nav_event_upp = NULL;  
  
    err = NavGetDefaultDialogOptions(&dialog_options);  
    if ( err == noErr )  
    {  
        dialog_options.dialogOptionFlags |= 0;  
        BlockMoveData( fileName, dialog_options.savedFileName, fileName[0] + 1 );  
        BlockMoveData( LMGetCurAppName(), dialog_options.clientName, LMGetCurAppName()[0] + 1 );  
        BlockMoveData( message, dialog_options.message, message[0] + 1 );  
  
        if ( NULL != user_event_proc )  
        {  
            nav_event_upp = NewNavEventUPP(user_event_proc);  
        }  
  
        // The file type and creator are the type and creator of the native file format.  
        // This is important because Nav Services allows the user to select a different  
        // format in the dialog box, and the translation manager does the translation  
        // from the native file type to the type selected by the user.  
        // If nav_event_upp is NULL, then the dialog is not movable or resizable.  
        err = NavPutFile(NULL, reply, &dialog_options, nav_event_upp, file_type, fileCreator, NULL);  
  
        if ( NULL != nav_event_upp )  
        {  
            DisposeNavEventUPP(nav_event_upp);  
        }  
  
        if (reply->validRecord)  
        {  
            // User saved  
            AEDesc resultDesc;  
            AEKeyword keyword;  
  
            // retrieve the returned selection:  
            err = AEGetNthDesc(&reply->selection, 1, typeFSS, &keyword, &resultDesc);  
            if (err == noErr) {  
                err = AEGetDescData(&resultDesc, file_fsspec, sizeof(FSSpec));  
            }  
        }  
        else  
        {  
            // User cancelled  
            err = userCanceledErr;  
        }  
    }  
  
    return err;  
}  
  
//  
// Call this routine after saving a document,  
// passing back the fileSpec and reply returned by SaveFileDialog.  
// This call performs any file tranlation needed and disposes the reply.  
  
OSStatus do_Nav_Complete_Save(const FSSpec *theSpec, NavReplyRecord* reply)  
{  
    #pragma unused(theSpec)  
    OSStatus err;  
  
    if (reply->validRecord)  
    {  
        err = NavCompleteSave(reply, kNavTranslateInPlace);  
    }  
  
    err = NavDisposeReply(reply);  
  
    return err;  
}  
  
//  
// Displays the save confirmation dialog anmd returns {ok, cancel, dontSaveChanges}  
  
short do_Nav_Confirm_Save( const unsigned char * documentName, Boolean quitting, NavEventProcPtr user_event_proc )  
{
```

```
// OSStatus theStatusErr = noErr;
OSStatus err = noErr;
NavAskSaveChangesResult reply = 0;
NavAskSaveChangesAction action = 0;
NavEventUPP nav_event_upp = NewNavEventUPP(user_event_proc);
NavDialogOptions dialog_options;
short result = cancel;

if (quitting)
    action = kNavSaveChangesQuittingApplication;
else
    action = kNavSaveChangesClosingDocument;

BlockMoveData(LMGetCurAppName(), dialog_options.clientName, LMGetCurAppName()[0]+1);
BlockMoveData(documentName, dialog_options.savedFileName, documentName[0]+1);

err = NavAskSaveChanges(    &dialog_options,
                           action,
                           &reply,
                           nav_event_upp,
                           NULL);

DisposeNavEventUPP(nav_event_upp);

// Map reply code to ok, cancel, dontSave
switch (reply)
{
    case kNavAskSaveChangesSave:
        result = ok;
        break;

    case kNavAskSaveChangesCancel:
        result = cancel;
        break;

    case kNavAskSaveChangesDontSave:
        result = dontSaveChanges;
        break;
}

return result;
}

//
// Callback to handle event passing between the navigation dialogs and the application
pascal void MyEventProc(const NavEventCallbackMessage callBackSelector, NavCBRecPtr callBackParms,
NavCallBackUserData callBackUD)
{
#pragma unused(callBackUD)

    if ( callBackSelector == kNavCBEvent )
    {
        switch (callBackParms->eventData.eventDataParms.event->what)
        {
            case updateEvt:
            case activateEvt:

                // remember not to adjust our menus while inside modal dialogs
                gInModalState = true;
                HandleEvent(callBackParms->eventData.eventDataParms.event);
                gInModalState = false;
                break;
        }
    }
}

//
//
#pragma mark -

//
// Allocates memory for a list of OSTypes and returns a handle to it.
static NavTypeListHandle NewOpenHandle( OSType app_signature, short num_types, OSType type_list[] )
{
    NavTypeListHandle type_list_hndl = NULL;

    if ( num_types > 0 )
    {
        type_list_hndl = (NavTypeListHandle)NewHandle( sizeof(NavTypeList) + ( num_types * sizeof(OSType) ) );

        if ( type_list_hndl != NULL )
        {
            (*type_list_hndl)->componentSignature = app_signature;
            (*type_list_hndl)->osTypeCount = num_types;
            BlockMoveData(type_list, (*type_list_hndl)->osType, num_types * sizeof(OSType));
        }
    }
}
```

```
    }  
    return type_list_hndl;  
}  
  
//  
static OSStatus SendOpenAE(AEDescList list)  
{  
    OSStatus      err;  
    AEDescDesc    theAddress;  
    AppleEvent    dummyReply;  
    AppleEvent    theEvent;  
  
    theAddress.descriptorType = typeNull;  
    theAddress.dataHandle     = NULL;  
  
    do {  
        ProcessSerialNumber psn;  
  
        err = GetCurrentProcess(&psn);  
        if ( err != noErr) break;  
  
        err = AECreatDesc(typeProcessSerialNumber, &psn, sizeof(ProcessSerialNumber), &theAddress);  
        if ( err != noErr) break;  
  
        dummyReply.descriptorType = typeNull;  
        dummyReply.dataHandle     = NULL;  
  
        err = AECreatAppleEvent(kCoreEventClass, kAEOpenDocuments, &theAddress, kAutoGenerateReturnID,  
kAnyTransactionID, &theEvent);  
        if ( err != noErr) break;  
  
        err = AEPutParamDesc(&theEvent, keyDirectObject, &list);  
        if ( err != noErr) break;  
  
        err = AESend(&theEvent, &dummyReply, kAEMWaitReply, kAENormalPriority, kAEDefaultTimeout, NULL, NULL);  
        if ( err != noErr) break;  
  
    } while (false);  
  
    return err;  
}  
  
//  
OSErr do_AEGetDescData(const AEDesc *desc, DescType *typeCode, void *dataBuffer, ByteCount maximumSize, ByteCount  
*actualSize)  
{  
    Handle h;  
    ByteCount dataSize;  
  
    *typeCode = desc->descriptorType;  
    h = (Handle)desc->dataHandle;  
    dataSize = GetHandleSize(h);  
  
    if (dataSize > maximumSize)  
        *actualSize = maximumSize;  
    else  
        *actualSize = dataSize;  
  
    BlockMoveData(*h, dataBuffer, *actualSize);  
    return noErr;  
}  
  
//  
#pragma mark -  
  
//  
// AddFileID creates a file ID reference for the  
// file specified by the FSSpec. It returns the  
// created file ID reference so that you can store  
// this reference for future use.  
  
OSErr  
do_Add_File_ID ( FSSpec *file, long *fileID )  
{  
    OSErr      err = NULL;  
    HParamBlkPtr    h = NULL;  
  
    h = (HParamBlkPtr)NewPtrClear(sizeof(HParamBlockRec));  
    if ( h != NULL)  
    {  
        h->fidParam.ioCompletion = nil;  
        h->fidParam.ioVRefNum = file->vRefNum;  
        h->fidParam.ioSrcDirID = file->parID;  
        h->fidParam.ioNamePtr = file->name;  
    }
```

```
err = PBCreateFileIDRefSync(h);

*fileID = h->fidParam.ioFileID;

DisposePtr ( (char *)h );
}

return ( err );
}

//
// GetFileID returns the File ID reference for a file
// where the File ID reference has previously been
// created (by calling PBCreateFileIDRef)

OSErr
do_Get_File_ID ( FSSpec *file, long *fileID )
{
    OSErr      err = NULL;
    CInfoPBPtr cInfo = NULL;

    cInfo = (CInfoPBPtr)NewPtrClear(sizeof(CInfoPBRec));
    if ( cInfo != NULL )
    {
        cInfo->hFileInfo.ioCompletion = nil;
        cInfo->hFileInfo.ioVRefNum = file->vRefNum;
        cInfo->hFileInfo.ioDirID = file->parID;
        cInfo->hFileInfo.ioNamePtr = file->name;
        cInfo->hFileInfo.ioFDirIndex = 0;

        err = PBGetCatInfoSync(cInfo);

        *fileID = cInfo->hFileInfo.ioDirID;

        DisposePtr ( (char *)cInfo );
    }

    return ( err );
}

//
OSErr do_Get_Maskerade_Sibling_FSSpec ( FSSpec *chosen_fsspec,
                                         FSSpec *sibling_fsspec,
                                         Boolean strip_extension,
                                         Boolean iterate_forward,
                                         int num_iterations )
{
    OSErr      err = noErr;
    char       full_name[256];
    unsigned char sibling_name[256];
    unsigned char test_name[256];
    unsigned char last_char;
    char       ext[256];
    int        i;

    // We need to clip the max number of file iterations to a reasonable number.
    if ( num_iterations < 1 )
        num_iterations = 1;
    else if ( num_iterations > 1000 )
        num_iterations = 1000;

    // Pull the name from the FSSpec and store it in a local string.

    if( strip_extension == true )
    {
        // If we're stripping the extension, we do the pstring to cstring thing,
        // parse the file name, and keep the extension-less part.
        do_p2c_strcpy ( full_name, chosen_fsspec->name );
        do_Split_DOS_Filename( full_name, NULL, NULL, NULL, NULL, (char *)sibling_name, ext );
        do_c2p_str( (char *)sibling_name );
    }
    else
    {
        // If ignoring the extension, we just put the full name
        // into the sibling name holder.
        do_p_strcpy ( sibling_name, chosen_fsspec->name );
    }

    // Search for a file extension
    // Strip it from the end and save both it and the remaining file name
    // Increment the file name
    // Reattach the extension before making an FSSpec

    for ( i = 0; i < num_iterations; i++ )
    {
        err = noErr;
    }
}
```

```
// First, we get the index of the last character. Thanks to array indices starting
// at '0', the last char of a pstring is at the array[length] position.
// We need to handle values of index == 0 or too small to decrement?
last_char = sibling_name[0];

// If it's a digit, we want to increment it by 1.
if ( isdigit ( sibling_name[last_char] ) )
{
    // If it's less than 9, we can just add 1 to it.
    if ( sibling_name[last_char] < '9' )
    {
        // We can make a name
        sibling_name[last_char] += 1;
    }
    // If it is a 9, we will need to wrap the number like an odometer.
    else
    {
        unsigned char    index;

        // We could do stuff in another order, but this order eliminates extra code.
        // We'll have to do some incrementing no matter what case we have, so we
        // handle the worst cases and save the incrementing until last.

        // First we handle the case where we have to roll the digits like an odometer.
        // While the indexed char is a 9 and the char to the left is a digit, we roll
        // the 9 to a 0 and move left one index.

        // We'll copy the last char to a local variable that we can decrement.
        index = last_char;

        while ( ( index > 1 ) && ( sibling_name[index] == '9' ) && isdigit ( sibling_name[index-1] ) )
        {
            // Since it's a 9, we roll it around to 0.
            sibling_name[index] = '0';

            // Then we move left one index and try again.
            index--;
        }

        // If we ended up with a 9 in the last attempted index, we'll need to add a character
        // to the name since the name will need another digit in it (for example, 99 to 100).
        if ( sibling_name[index] == '9' )
        {
            // First we have to check if there's enough room for another character.
            if ( sibling_name[0] < kMaxNumFileNameCharacters )
            {
                // We roll the char to a 1, then add a 0 to the end.
                // Then we make the string one char longer.
                // We can make a name
                sibling_name[index] = '1';
                sibling_name[last_char + 1] = '0';
                sibling_name[0] += 1;
            }
            else
            {
                // Would be too many characters in sibling filename.
                do_p_strcpy ( sibling_fsspec->name, sibling_name );
                err = errFSNameTooLong;
                break;
            }
        }
        // Since we don't have a 9, we can just increment it.
        else
        {
            // We can make a name
            sibling_name[index] += 1;
        }
    }
}

// If it's a letter, we want to increment it to the next letter.
else if ( isalpha ( sibling_name[last_char] ) )
{
    // If it's a 'z' or 'Z', we don't want to do anything.
    if ( ( sibling_name[last_char] == 'z' ) || ( sibling_name[last_char] == 'Z' ) )
    {
        // At end of filename iteration.
        do_p_strcpy ( sibling_fsspec->name, sibling_name );
        err = errFSNoMoreItems;
        break;
    }
    // Otherwise, we step the letter to the next letter.
    else
    {
        sibling_name[last_char] += 1;
    }
}
else
```



```
{
    // If it's not a digit or character, we bail since
    // we can't make sibling filename from it.
    do_p_strcpy ( sibling_fsspec->name, sibling_name );
    err = errFSBadForkName;
    break;
}

// If we're still here, we have a possible file name.
do_p_strcpy( test_name, sibling_name );

if( strip_extension == true )
{
    do_c2p_strcat( test_name, ext );
}
else
{
    // We don't need to do anything else.
}

// Now we check to see if it exists.
err = FSMakeFSSpec ( chosen_fsspec->vRefNum, chosen_fsspec->parID, test_name, sibling_fsspec );
if ( err == noErr )
{
    // If there's no error, we got a file or directory.
    // We need to check which of the two it is.

//      FileInfo      file_info;
//      Boolean        is_dir = false;
//      long           dir_id = 0L;

//      err = FSpGetFInfo ( sibling_fsspec, &file_info );
//      err = FSpGetDirectoryID ( sibling_fsspec, &dir_id, &is_dir );
//      if ( err == noErr )
//      {
//          if ( is_dir == false )
//          {
//              // It's a file, so we can bail.
//              break;
//          }
//          else
//          {
//              // It's a directory, so we need to ignore it and
//              // continue searching. We'll flag the error as
//              // file not found in case this was the last
//              // file/folder of the iteration.
//              err = fnfErr;
//              continue;
//          }
//      }
//      else
//      {
//          // Something went wrong, so we bail.
//          // We leave the FSpGetDirectoryID() error as the return value.
//          break;
//      }
//      else if ( err == fnfErr )
//      {
//          // If the specified file doesn't exist, we iterate again.
//          continue;
//      }
//      else
//      {
//          // Something went wrong making the FSSpec, so we bail.
//          break;
//      }
//      }

    return ( err );
}

//
//
#pragma mark -
//
//
OSErr do_Get_App_FSSpec ( FSSpec *app_fsspec )
{
    OSErr                err = noErr;
    ProcessSerialNumber   psn;
    ProcessInfoRec        info_rec;
    //Str255               app_name;

    // We have to initialize these fields! If we don't,
    // GetProcessInformation will crash us. Even though the
    // process name is returned, we won't use it.
```

```
info_rec.processInfoLength = sizeof( info_rec );
info_rec.processName = NULL;
info_rec.processAppSpec = app_fsspec;

GetCurrentProcess ( &psn );
err = GetProcessInformation ( &psn, &info_rec );

if ( err == noErr )
    return ( noErr );
else
    return ( paramErr );
}

//
// Pass in the FSSpec of an object plus the name of a sibling object,
// and you'll get a FSSpec for the sibling object if it exists.
// A Boolean indicates whether or not the object was a folder.

OSErr
do_Assert_Sibling_File_FSSpec ( FSSpec *known_fsspec, unsigned char *sibling_name, FSSpec *sibling_fsspec, Boolean
*is_dir )
{
    OSErr      err = noErr;
    long       parent_id = 0;

    err = GetParentID ( known_fsspec->vRefNum, known_fsspec->parID, known_fsspec->name, &parent_id );
    if ( err == noErr )
    {
        err = FSMakeFSSpec ( known_fsspec->vRefNum, parent_id, sibling_name, sibling_fsspec );
        if ( err == noErr )
        {
            // We'll just reuse the parent_id variable since we don't need it anymore
            // and don't care what's returned.
            err = FSpGetDirectoryID ( sibling_fsspec, &parent_id, is_dir );
        }
        else if ( fnfErr == err )
        {
            *is_dir = false;
        }
    }

    return ( err );
}

//
// Pass in the FSSpec of a folder plus the name of an object
// assumed to be inside the folder, and you'll get a FSSpec
// for the object, whether or not it exists. A Boolean indicates
// whether or not the object was a folder.

OSErr do_Assert_Child_File_FSSpec ( FSSpec *parent_fsspec, unsigned char *child_name, FSSpec *child_fsspec,
Boolean *is_dir )
{
    OSErr      err = noErr;
    long       parent_id = 0;

    // If it's not a folder, it can't have children.
    // If the FSSpec has no name, we could just use the parID field,
    // but this method always gives us the folder's correct id.
    err = FSpGetDirectoryID ( parent_fsspec, &parent_id, is_dir );
    if ( err == noErr && *is_dir == true )
    {
        err = FSMakeFSSpec ( parent_fsspec->vRefNum, parent_id, child_name, child_fsspec );
        if ( err == noErr )
        {
            // We'll just reuse the parent_id variable since we
            // don't need it anymore and don't care what's returned.
            err = FSpGetDirectoryID ( child_fsspec, &parent_id, is_dir );
        }
        else if ( fnfErr == err )
        {
            *is_dir = false;
        }
    }

    return ( err );
}

//
#pragma mark -

//
// Pass in the FSSpec of an object, and you'll get a FSSpec
// for the object's parent folder if it exists.

OSErr
do_Get_Parent_Folder ( FSSpec *child_fsspec, FSSpec *parent_fsspec )
```

```
{
OSErr          err = noErr;
long           parent_id = 0;

    // No matter what kind of FSSpec we have, we pass in its info and get back the object's
    // parent folder id if it has one.
    err = GetParentID ( child_fsspec->vRefNum, child_fsspec->parID, child_fsspec->name, &parent_id );
    if ( err == noErr )
    {
        err = FSMakeFSSpec ( child_fsspec->vRefNum, parent_id, "\p", parent_fsspec );
    }

    return ( err );
}

//
// So we don't have to error check later, this WILL return a
// valid ID for a folder. It may not be the folder you want,
// but the folder returned will be there.

OSErr do_Assert_Sibling_Folder ( FSSpec *known_fsspec, unsigned char *sibling_name,
                                FSSpec *sibling_fsspec, Boolean *is_dir )
{
    long         dir_id = 0L;
    OSErr        err = noErr;
    unsigned char default_name[] = "\puntitled folder";
    char         error_string[256];

    // Check to see if we got a NULL pointer or a blank folder name.
    // If so, we go ahead and set a default folder name.
    if ( sibling_name == NULL || *sibling_name == 0 )
    {
        // We set the pointer to the local pstring.
        sibling_name = default_name;
    }

    // We copy the name to a local cstring so we can use it in the debugging prints.
    do_p2c_strcpy(error_string,sibling_name);

    // We make an FSSpec with the info and see if the item already exists.
    err = FSMakeFSSpec ( known_fsspec->vRefNum, known_fsspec->parID, sibling_name, sibling_fsspec );
    if ( err == fnfErr )
    {
        // If not, we try to create a new folder.
        err = FSpDirCreate ( sibling_fsspec, smSystemScript, &dir_id );
        if ( err == noErr )
        {
            DEBUG_VAR_PRINT("\'%s\' folder successfully created",error_string);
        }
        else
        {
            DEBUG_VAR_PRINT("\'%s\' folder could not be created",error_string);
            DEBUG_EXTRA_VAR_PRINT("because error %d occurred",err);
        }
    }
    else if ( err == noErr )
    {
        // If the FSSpec already exists, we check to see if it's a folder or not.
        err = FSpGetDirectoryID ( sibling_fsspec, &dir_id, is_dir );
        if ( err == noErr )
        {
            DEBUG_VAR_PRINT("\'%s\' folder already exists",error_string);
        }
        else
        {
            DEBUG_VAR_PRINT("Could not determine if \''%s\' is a folder",error_string);
            DEBUG_EXTRA_VAR_PRINT("because error %d occurred getting the id of the folder",err);
        }
    }

    return ( err );
}

//
#pragma mark -

//
// Checks a string for wildcard characters ('?' and '*')
// Arguments 1 - String to check
// Returns: True_if string contains wildcards, else False_
// Side Effects: None

Boolean do_Filename_Has_Wildcard_Chars( char *pname )
{
    if ( NULL != strchr(pname, '*') || NULL != strchr(pname, '?') )
        return true;
    else

```

```
        return false;
    }

//
// Splits file specifications into component parts. Similar to
// compiler-specific fnsplit() or _splitpath().
// Arguments      1 - Original file specification
//                2 - Buffer to receive drive spec
//                3 - Buffer to receive drive/path spec
//                4 - Buffer to receive path spec
//                5 - Buffer to receive name.ext spec
//                6 - Buffer to receive name spec
//                7 - Buffer to receive ext spec
// Returns: Bit map as follows (see defines in RBS.H):
//          Extension_ - File spec included extension
//          Filename_  - File spec did not end in '\'
//          Directory_ - File spec included a path
//          Drive_     - File spec included a drive spec
//          Wildname_  - File name included wildcards (*.?)
//          Wildpath_  - File path included wildcards (*.?)
// Side Effects: Calls unix2dos() to convert '/' to '\'
// Notes: Passing NULL in arguments 2-7 causes fnsplit() to
// not save the corresponding portion of the path.

int do_Split_DOS_Filename( char *spec,      // Original file spec
                          char *drive,     // Drive spec
                          char *pname,     // Path w/ drive spec
                          char *path,      // Path spec
                          char *fname,     // File name + extension
                          char *name,      // File name
                          char *ext)       // File extension
{
    int ret_code = 0;
    char *d = spec, *p, *e;

    // First we call unix2dos() to convert '/' to '\'
    do_UNIX_To_DOS_Path(spec);

    // Check to see if there's a drive specified
    if (':' == spec[1])
    {
        if (drive)
            strncpy(drive, spec, 2);
        drive[2] = NULL;
        d += 2;
        ret_code |= Drive_;
    }
    else
    {
        if ( drive != NULL )
            *drive = NULL;
    }

    // Now we look for the last '\'
    if ( NULL != ( p = strrchr(d, '\\') ) )
    {
        char ch;

        ch = *(++p);
        *p = NULL;

        if ( path != NULL )
            strcpy(path, d);

        if ( pname != NULL )
            strcpy(pname, spec);

        if ( do_Filename_Has_Wildcard_Chars(d) != NULL )
            ret_code |= Wildpath_;

        *p = ch;
        ret_code |= Directory_;
    }
    else
    {
        if ( path != NULL )
            *path = NULL;

        if ( pname != NULL )
        {
            if ( drive != NULL )
                strcpy(pname, drive);
            else
                *pname = NULL;
        }

        p = d;
    }
}
```

```
//
// Creates file specification from component parts. Similar to
// compiler-specific fnmerge() or _makepath().
// Arguments      1 - Original file specification
//               2 - Buffer to receive drive spec
//               3 - Buffer to receive drive/path spec
//               4 - Buffer to receive path spec
//               5 - Buffer to receive name.ext spec
//               6 - Buffer to receive name spec
//               7 - Buffer to receive ext spec
//
// Returns: Reassembled name
// Side Effects: None
```

```
{
    *spec = NULL;

    if ( (pname != NULL) && (*pname != NULL) )
    {
        strcpy(spec, pname);
    }
    else
    {
        if ( (drive != NULL) && (*drive != NULL) )
            strcpy(spec, drive);

        if ( (path != NULL) && (*path != NULL) )
            strcpy(spec, path);
    }

    do_UNIX_To_DOS_Path(spec);

    if ( (*spec != NULL) && ('\\' != LAST_CHAR(spec)) && (':' != LAST_CHAR(spec)) )
        strcat(spec, "\\");

    if ( (fname != NULL) && (*fname != NULL) )
        strcat(spec, fname);
    else
    {
        if ( (name != NULL) && (*name != NULL) )
            strcat(spec, name);
        else
            return spec;

        if ( (ext != NULL) && (*ext != NULL) )
        {
            if ( '.' != *ext )
                strcat(spec, ".");

            strcat(spec, ext);
        }
    }

    return do_strupr(spec);
}
```

```
//
#pragma mark -
//
```

```
char *do_UNIX_To_DOS_Path(char *path)
{
    char *p;

    if( path == NULL )
        return( NULL );

    for (p = path; *p; ++p)
    {
        if ('/' == *p)
            *p = '\\';
    }

    return path;
}
```

```
//
char *do_DOS_To_UNIX_Path(char *path)
{
    char *p;

    if( path == NULL )
        return( NULL );

    for (p = path; *p; ++p)
    {
        if ('\\' == *p)
            *p = '/';
    }

    return path;
}
```

```
//
char *do_MAC_To_UNIX_Path(char *path)
{
    char *p;

    if( path == NULL )
        return( NULL );
}
```

```
    for (p = path; *p; ++p)
    {
        if (':' == *p)
            *p = '/';
    }
    return path;
}

//
char *do_UNIX_To_MAC_Path(char *path)
{
    char *p;

    if( path == NULL )
        return( NULL );

    for (p = path; *p; ++p)
    {
        if ( '/' == *p )
            *p = ':';
    }
    return path;
}
```

```
//  
//  
//  
#ifndef __my_gestalts__  
#define __my_gestalts__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Appearance.h>  
#include <OpenTransport.h>  
#include <Gestalt.h>  
#include <TextUtils.h>  
#include <Navigation.h>  
#endif  
#endif  
  
#include <stdlib.h>  
  
#include "my_alerts.h"  
#include "my_macros.h"  
#include "my_utilities.h"  
  
#ifdef cplusplus  
extern "C" {  
#endif  
  
#define kAlertStringGestaltManagerError "\pThe Gestalt Manager was unable to check your system for necessary  
features."  
#define kAlertString_gestaltUnknownErr "\pGestalt was unable to obtain a response from the system."  
#define kAlertString_gestaltUndefSelectorErr "\pAn undefined selector was passed to Gestalt."  
#define kAlertString_gestaltDupSelectorErr "\pAn attempt was made to add a Gestalt entry that already exists."  
#define kAlertString_gestaltLocationErr "\pThe Gestalt function pointer wasn't in the system heap."  
  
#define kAlertStringNoSystemVersionError "\pThis application requires a newer version of system software."  
// #define kAlertStringNoSystemVersionExplanation "\pPlease make sure that an appropriate system version is  
installed."  
#define kAlertStringNoSystemVersionExplanation "\pMake sure that a suitable version of system software is  
installed by choosing \"About This Mac\" from the Apple menu or by using Apple System Profiler."  
  
#define kAlertStringNoCarbonVersionError "\pThis application requires a newer version of CarbonLib."  
// #define kAlertStringNoCarbonVersionExplanation "\pPlease make sure that an appropriate CarbonLib is installed."  
#define kAlertStringNoCarbonVersionExplanation "\pUsing Apple System Profiler, check your extensions (OS 9) or  
frameworks (OS X) and make sure that a suitable version of CarbonLib is installed."  
  
#define kAlertStringNoAppearanceManagerError "\pThis application requires the Appearance Manager."  
#define kAlertStringNoAppearanceManager101Error "\pThis application requires the Appearance Manager 1.0.1 or  
newer."  
#define kAlertStringNoAppearanceManagerPreOS8Explanation "\pYou are running a system older than Mac OS 8.0. Please  
install Appearance 1.0.2 or newer for System 7.1 thru 7.6."  
#define kAlertStringNoAppearanceManagerExplanation "\pPlease make sure that Appearance 1.0.1 or newer is  
installed."  
  
#define kAlertStringNoPlatinumAppearanceError "\pSome dialogs and controls may have a pre-Appearance Manager  
look."  
#define kAlertStringNoPlatinumAppearanceExplanation "\pSystem-Wide Platinum appearance is turned off in the  
Appearance control panel."  
  
#define kAlertStringNoColorSyncError "\pThis application requires a newer version of ColorSync."  
// #define kAlertStringNoColorSyncExplanation "\pPlease consult the application's system requirements for the  
minimum version."  
#define kAlertStringNoColorSyncExplanation "\pUsing Apple System Profiler, check your extensions (OS 9) or  
frameworks (OS X) and make sure that a suitable version of ColorSync is installed."  
  
#define kAlertStringNoColorQuickDrawError "\pThis application requires Color QuickDraw 1.3 or newer."  
#define kAlertStringNoColorQuickDrawExplanation "\pPlease make sure that System 7 or newer is installed."  
  
#define kAlertStringNoQuickTimeError "\pThis application requires QuickTime."  
#define kAlertStringPre25QuickTimeErr "\pPlease make sure that version 2.5 or newer is installed."  
#define kAlertStringPre30QuickTimeErr "\pPlease make sure that version 3.0 or newer is installed."  
  
#define kAlertStringPreReleaseQuickTimeError "\pSome features provided by QuickTime may not be stable."  
#define kAlertStringPreReleaseQuickTimeExplanation "\pA pre-release version of QuickTime is installed."
```



```
#define kAlertStringNoQuickTimePowerPlugError "\pThe PowerPC QuickTime glue library is not present."
#define kAlertStringNoQuickTimePowerPlugExplanation "\pPlease make sure that the QuickTime PowerPlug is
installed."

#define kAlertStringNoOpenTransportError "\pThis application requires Open Transport."
#define kAlertStringNoOpenTransportExplanation "\pPlease make sure that Open Transport 1.2 or newer is installed."

#define kAlertStringNoAppleTalkError "\pAppleTalk is not currently available."
#define kAlertStringNoAppleTalkExplanation "\pAppleTalk is required for a site license version of this program."

#define kAlertStringNoDragManagerError "\pThis application requires the Drag Manager."
#define kAlertStringNoDragManagerExplanation "\pPlease make sure that the Drag Manager is properly installed."

#define kAlertStringNoQD3DLError "\pThis application requires QuickDraw 3D."
#define kAlertStringNoQD3DLEExplanation "\pPlease make sure that QuickDraw 3D is properly installed (PowerPC only)."
#define kAlertStringNoQD3DVersionError "\pThis application requires QuickDraw 3D version 1.5.4 or later."

#define kAlertStringNoQD3DViewerError "\pThis application requires the QuickDraw 3D viewer."
#define kAlertStringNoQD3DViewerExplanation "\pPlease make sure that the QuickDraw 3D Viewer extension is properly
installed (PowerPC only)."

#define kAlertStringNoAppleEventsError "\pThis application requires support for Apple Events."
#define kAlertStringNoAppleEventsExplanation "\pPlease make sure that the system was properly installed."

enum
{
    kmyStopAlertALRTID = 1100
};

/*
enum {
    // Classic presence and features
    gestaltMacOSCompatibilityBoxAttr = FOUR_CHAR_CODE('bbox'),
    // True if running under the Classic
    gestaltMacOSCompatibilityBoxPresent = 0,
    // True if Classic serial support is implemented.
    gestaltMacOSCompatibilityBoxHasSerial = 1,
    // True if we're Boxless (screen shared with Carbon/Cocoa)
    gestaltMacOSCompatibilityBoxless = 2
};
*/

void do_Init_Toolbox ( void );
void do_Init_Memory ( int );
OSStatus do_Init_Help ( void );

void * do_Get_Routine_Pointer( const unsigned char *, const unsigned char * );

int do_Check_For_System_Version ( short );
int do_Check_For_Carbon_Version ( short );
int do_Check_For_Appearance_Manager ( short );
Boolean do_Check_If_Running_On_Classic( void );
Boolean do_Check_If_Running_On_Carbon_X( void );
Boolean do_Check_For_Aqua_Menus ( void );
int do_Check_For_ColorSync ( short );
int do_Check_For_Color_QuickDraw ( short );
int do_Check_For_Display_Manager ( short );
int do_Check_For_QuickTime ( short );
int do_Check_For_Open_Transport ( short );
int do_Check_For_Drag_Manager ( short );
int do_Check_For_Apple_Events ( short );
Boolean do_Check_For_Nav_Services ( void );
int do_Check_For_QD3D ( short );
int do_Check_For_QD3D_Viewer ( short );

void do_Bail_From_Gestalt (OSErr);

#ifdef __cplusplus
}
#endif

#endif /* __my_gestalts__ */
```

```
//  
// _____ ©1998-2001 bergdesign inc.  
//  
// _____  
// 2002-0-16 Added version numbers to the error messages of the gestalt checking routines  
//  
  
#include "my_gestalts.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
// _____  
  
void do_Init_Toolbox ()  
{  
    // These are the standard pre-Carbon initialization routines.  
    #if !TARGET_API_MAC_CARBON  
    // InitGraf( GetQDGlobalsThePort() );  
    InitGraf( &qd.thePort );  
    InitFonts();  
    InitWindows();  
    InitFloatingWindows();  
    InitMenus();  
    TEInit();  
    InitDialogs( NULL );  
    #endif  
  
    // Initializes the standard arrow cursor. Since the system  
    // sets the cursor to the beachball on application launch,  
    // it will stay that way until InitCursor() is called.  
    InitCursor ();  
  
    // This initializes the animated cursor with resource ID 0.  
    // We can call SpinCursor(0) to get it.  
    // This only works with the MPW library linked in.  
    // InitCursorCtl ( NULL );  
  
    // This empties the event queue completely before starting.  
    FlushEvents( everyEvent, 0 );  
}  
  
// _____  
  
void do_Init_Memory ( int num_blocks )  
{  
    int i;  
  
    // MaxApplZone() immediately expands the application heap  
    // to its limit. If you don't do this, the Memory Manager  
    // gradually expands your heap as memory needs increase  
    // which results in significant heap fragmentation.  
    #if !TARGET_API_MAC_CARBON  
    MaxApplZone ();  
    #endif  
  
    // MoreMasters() allocates an additional block of memory for 64 master pointers in  
    // your application heap, and 32 master pointers in the system heap. You should  
    // call it as many times as you need additional pointers.  
    for ( i = 0; i < num_blocks; i++ )  
    {  
        MoreMasters();  
    }  
}  
  
// _____  
  
OSStatus do_Init_Help ( void )  
{  
    CFBundleRef myAppsBundle;  
    CFURLRef myBundleURL;  
    FSRef myBundleRef;  
    OSStatus err;  
  
    /* set up a known state */  
    myAppsBundle = NULL;  
    myBundleURL = NULL;  
  
    /* Get our application's main bundle from Core Foundation */  
    myAppsBundle = CFBundleGetMainBundle();  
    if (myAppsBundle == NULL)  
    {  
        err = fnfErr;  
        goto bail;  
    }  
  
    /* retrieve the URL to our bundle */  
    myBundleURL = CFBundleCopyBundleURL(myAppsBundle);
```

```
    if (myBundleURL == nil)
    {
        err = fnfErr;
        goto bail;
    }

    /* convert the URL to a FSRef */
    if ( !CFURLGetFSRef(myBundleURL, &myBundleRef) )
    {
        err = fnfErr;
        goto bail;
    }

    /* register our application's help book */
    err = AHRRegisterHelpBook(&myBundleRef);
    if (err != noErr)
        goto bail;

    /* done */
    CFRelease(myBundleURL);
    return noErr;

bail:

    if (myBundleURL != NULL)
        CFRelease(myBundleURL);

    return err;
}

// _____

#pragma mark -

// _____

// How do I call a routine in InterfaceLib from a Carbon application?
// This example checks the existence of the routine, NSetTrapAddress() within InterfaceLib, and if available calls
// it.
//
typedef pascal void (*NSetTrapAddressProcPtr) ((UniversalProcPtr)trapAddr, UInt16 trapNum, TrapType tTyp)

CFragConnectionID connID = kInvalidID;

OSErr err = GetSharedLibrary( "pInterfaceLib", kCompiledCFragArch, kReferenceCFrag, &connID, NULL, NULL );
if ( err == noErr )
{
    NSetTrapAddressProcPtr myNsetTrapAddressProcPtr = NULL;

    err = FindSymbol( connID, "pNsetTrapAddress", (Ptr *) &myNsetTrapAddressProcPtr, NULL );
    if ( err == noErr )
    {
        // Routine is available!
        (*myNsetTrapAddressProcPtr) ( trapAddr, trapNum, tTyp );
    }
}
*/

void * do_Get_Routine_Pointer( const unsigned char *lib_name, const unsigned char *func_name )
{
    CFragConnectionID connID = kInvalidID;
    void *func_ptr = NULL;

    OSErr err = GetSharedLibrary( lib_name, kCompiledCFragArch, kReferenceCFrag, &connID, NULL, NULL );
    if ( err == noErr )
    {
        err = FindSymbol( connID, func_name, (Ptr *) &func_ptr, NULL );
        if ( err != noErr ) // Symbol was not available
            func_ptr = NULL; // Set ptr to NULL since we don't know the state of it
    }

    return( func_ptr );
}

// _____

#pragma mark -

// _____

int do_Check_For_System_Version ( short version )
{
    OSErr      err = noErr;
    Boolean    is_present = false;
    long       result = 0L;

    err = Gestalt ( gestaltSystemVersion, &result );
    if ( err == noErr )
    {

```

```
DEBUG_VAR_PRINT("System version is %hx", HIWORD ( result ));
DEBUG_VAR_PRINT("System release is %hx", LOWORD ( result ));

if ( LOWORD ( result ) >= version )
{
    is_present = true;
}
else
{
    Str255 error_txt;
    Str255 explain_txt;

    error_txt[0] = 0;
    explain_txt[0] = 0;

    do_p_strcat( error_txt, "\pThis application requires Mac OS " );
    do_p_strBCDcat ( error_txt, version );
    do_p_strcat( error_txt, "\p or newer." );

    do_p_strcat( explain_txt, "\pMac OS " );
    do_p_strBCDcat ( explain_txt, LOWORD ( result ) );
    do_p_strcat( explain_txt, "\p is installed. You will need to update your system software to run this
application." );

    do_One_Button_Alert ( kAlertStopAlert,
                        error_txt,
                        explain_txt,
                        "\pOK" );

    is_present = false;
}
else
{
    do_Bail_From_Gestalt ( err );
}

return ( is_present );
}

//
int do_Check_For_Carbon_Version( short version )
{
    OSErr      err = noErr;
    Boolean    is_present = false;
    long       result = 0L;

    err = Gestalt ( gestaltCarbonVersion, &result );
    if ( err == noErr )
    {
        DEBUG_VAR_PRINT("Carbon version is %hx", HIWORD ( result ));
        DEBUG_VAR_PRINT("Carbon release is %hx", LOWORD ( result ));

        if ( LOWORD ( result ) >= version )
        {
            is_present = true;
        }
        else
        {
            Str255 error_txt;
            Str255 explain_txt;

            error_txt[0] = 0;
            explain_txt[0] = 0;

            do_p_strcat( error_txt, "\pThis application requires Carbon " );
            do_p_strBCDcat ( error_txt, version );
            do_p_strcat( error_txt, "\p or newer." );

            do_p_strcat( explain_txt, "\pCarbon " );
            do_p_strBCDcat ( explain_txt, LOWORD ( result ) );
            do_p_strcat( explain_txt, "\p is installed. You will need to install a newer version of CarbonLib (OS 9
or Carbon.framework (OS X) to run this application." );

            do_One_Button_Alert ( kAlertStopAlert,
                                error_txt,
                                explain_txt,
                                "\pOK" );

            is_present = false;
        }
    }
    else
    {
        do_Bail_From_Gestalt ( err );
    }
}
```

```
    return ( is_present );
}

//
int do_Check_For_Appearance_Manager ( short version )
{
    OSErr          err = noErr;
    Boolean        is_present = false;
    long           Attr_result = 0L;
    long           Version_result = 0L;
    //Str255        string_0, string_1, string_2, string_3;

    // Do we have any knowledge about the appearance manager?
    err = Gestalt ( gestaltAppearanceAttr, &Attr_result );
    if ( err == noErr )
    {
        // Ok, we didn't get an error so the selector exists and we know about
        // the appearance manager. Now we need to check and see if it is installed.
        if ( do_Is_Bit_Set ( Attr_result, gestaltAppearanceExists ) )
        {
            is_present = true;

            // Yes it is installed, but what version is it?
            err = Gestalt ( gestaltAppearanceVersion, &Version_result );

            // If the gestaltAppearanceVersion selector is undefined,
            // we're running 1.0 which isn't good enough.
            if ( err == gestaltUndefSelectorErr )
            {
                is_present = false;

                do_One_Button_Alert ( kAlertStopAlert,
                                    kAlertStringNoAppearanceManager101Error,
                                    kAlertStringNoAppearanceManagerExplanation,
                                    "\pOK" );
            }
            else if ( err != noErr )
            {
                do_Bail_From_Gestalt ( err );
            }
            else
            {
                // Ok, we know we're running at least 1.0, but which specific version is it?
                // The version is a word, with the upper byte being the major revision number
                // and the lower byte being the minor revision number
                if ( LOWORD ( Version_result ) < version )
                {
                    Str255 error_txt;
                    Str255 explain_txt;

                    error_txt[0] = 0;
                    explain_txt[0] = 0;

                    do_p_strcat( error_txt, "\pThis application requires AppearanceManager " );
                    do_p_strBCDcat ( error_txt, version );
                    do_p_strcat( error_txt, "\p or newer." );

                    do_p_strcat( explain_txt, "\pAppearanceManager " );
                    do_p_strBCDcat ( explain_txt, LOWORD ( Version_result ) );
                    do_p_strcat( explain_txt, "\p is installed. You will need to install a newer version of
AppearanceManager to run this application." );

                    do_One_Button_Alert ( kAlertStopAlert,
                                        error_txt,
                                        explain_txt,
                                        "\pOK" );

                    is_present = false;

                    // DEBUG_VAR_PRINT("Lo Word = %x", LOWORD ( Version_result ));
                    // DEBUG_VAR_PRINT("Hi Word = %x", HIWORD ( Version_result ));
                }
            }
            else
            {
                is_present = true;

                // Ok, we have a sufficient version running, but we should check to see if
                // is compatibility mode is turned on and warn the user that some things might
                // not look right if it's not turned on.
                if ( do_Is_Bit_Set ( Attr_result, gestaltAppearanceCompatMode ) )
                {
                    do_One_Button_Alert ( kAlertNoteAlert,
                                        kAlertStringNoPlatinumAppearanceError,
                                        kAlertStringNoPlatinumAppearanceExplanation,
                                        "\pOK" );
                }
            }
        }
    }
}
```

```
    }  
// No, the appearance manager is not installed.  
// We have to call an old style alert to notify the user.  
else  
{  
    is_present = false;  
  
//    sprintf ( (char *)string_0, (const char *)kAlertStringNoAppearanceManagerError );  
//    do_c2p_str ( (char *)string_0 );  
//    sprintf ( (char *)string_1, (const char *)kAlertStringNoAppearanceManagerExplanation );  
//    do_c2p_str ( (char *)string_1 );  
//    sprintf ( (char *)string_2, "" );  
//    do_c2p_str ( (char *)string_2 );  
//    sprintf ( (char *)string_3, "" );  
//    do_c2p_str ( (char *)string_3 );  
  
//    ParamText ( string_0, string_1, string_2, string_3 );  
//    ParamText ( kAlertStringNoAppearanceManagerError, kAlertStringNoAppearanceManagerExplanation, "\p", "\p  
);  
    StopAlert ( kmyStopAlertALRTID, NULL );  
}  
// If the gestaltAppearanceAttr selector is undefined,  
// we're on system 7.x with no knowledge of the appearance manager  
else if ( err == gestaltUndefSelectorErr )  
{  
    is_present = false;  
  
    do_One_Button_Alert ( kAlertStopAlert,  
                          kAlertStringNoAppearanceManagerError,  
                          kAlertStringNoAppearanceManagerPreOS8Explanation,  
                          "\pOK" );  
}  
else  
{  
    do_Bail_From_Gestalt ( err );  
}  
  
if ( is_present )  
{  
    #if !TARGET_API_MAC_CARBON  
        err = RegisterAppearanceClient ();  
        if ( err != noErr )  
        {  
            is_present = false;  
            DEBUG_VAR_PRINT("Could not register the application as Appearance Manager savvy. Error %d.", err );  
        }  
    #endif  
}  
  
return ( is_present );  
}  
  
//  
// Q: My application does things that just won't work in the Classic  
// environment. How do I detect the Classic environment so that I can  
// tell the user why certain functionality has been disabled?  
// A: You can detect the Classic environment using Gestalt, as shown below.  
  
Boolean do_Check_If_Running_On_Classic( void )  
{  
    OSErr      err = noErr;  
    UInt32     response = 0;  
    Boolean    running_on_classic = false;  
  
    err = Gestalt(gestaltMacOSCompatibilityBoxAttr, (SInt32 *)&response);  
    if ( err == noErr )  
    {  
        running_on_classic = do_Is_Bit_Set( response, gestaltMacOSCompatibilityBoxPresent );  
    }  
  
    return( running_on_classic );  
}  
  
//  
// Q: My Carbon application does things on traditional Mac OS that just  
// won't work on Mac OS X. How do I detect that I'm running on Mac OS X  
// so that I can do things the right way on that platform?  
// A: Apple Developer Tech Support (DTS) recommends that you test for  
// specific functionality rather than for an entire platform. For  
// example, if your application needs access to non-Carbon APIs on  
// traditional Mac OS, we recommend that you access those APIs using  
// GetSharedLibrary and FindSymbol; if the GetSharedLibrary call fails,  
// you don't have access to the functionality, regardless of the platform.  
// On the other hand, we recognize that in some cases there is no  
// convenient functional test, and only a platform test is possible. In  
// such cases, your Carbon application can detect whether it is running
```

// on Mac OS X using Gestalt, as shown in Listing 2.

```
Boolean do_Check_If_Running_On_Carbon_X( void )
{
    OSErr      err = noErr;
    Boolean    is_present = false;
    long       result = 0L;
```

```
    err = Gestalt ( gestaltSystemVersion, &result );
    if ( err == noErr )
    {
        DEBUG_VAR_PRINT("System version is %hx", HIWORD ( result ));
        DEBUG_VAR_PRINT("System release is %hx", LOWORD ( result ));

        if ( LOWORD ( result ) >= 0x1000 )
            is_present = true;
        else
            is_present = false;
    }
    else
    {
        do_Bail_From_Gestalt ( err );
    }

    return ( is_present );
}
```

// _____

```
Boolean do_Check_For_Aqua_Menus( void )
{
    OSErr      err = noErr;
    long       result = 0L;
    Boolean    is_present = false;
```

```
    err = Gestalt ( gestaltMenuMgrAttr, &result );
    if( err == noErr )
    {
        if ( do_Is_Bit_Set( result, gestaltMenuMgrPresentBit )
            && do_Is_Bit_Set( result, gestaltMenuMgrAquaLayoutBit ) )
        {
            is_present = true;
        }
    }

    return( is_present );
}
```

// _____

```
int do_Check_For_ColorSync ( short version )
{
    OSErr      err = noErr;
    Boolean    is_present = false;
    long       result = 0L;
```

```
    err = Gestalt ( gestaltColorMatchingVersion, &result );
    if ( err == noErr )
    {
        if ( result < version )
        {
            Str255 error_txt;
            Str255 explain_txt;

            error_txt[0] = 0;
            explain_txt[0] = 0;

            do_p_strcat( error_txt, "\pThis application requires ColorSync " );
            do_p_strBCDcat ( error_txt, version );
            do_p_strcat( error_txt, "\p or newer." );

            do_p_strcat( explain_txt, "\pColorSync " );
            do_p_strBCDcat ( explain_txt, LOWORD ( result ) );
            do_p_strcat( explain_txt, "\p is installed. You will need to install a newer version of ColorSync to run this application." );

            do_One_Button_Alert ( kAlertStopAlert,
                                error_txt,
                                explain_txt,
                                "\pOK" );

            is_present = false;
        }
        else
        {
            is_present = true;
        }
    }
}
```

```
    else
    {
        do_Bail_From_Gestalt ( err );
    }

    return ( is_present );
}

//
int do_Check_For_Color_QuickDraw ( short version )
{
    OSErr      err = noErr;
    Boolean     is_present = false;
    long        result = 0L;

    err = Gestalt ( gestaltQuickdrawVersion, &result );
    if ( err == noErr )
    {
        if ( result < gestalt32BitQD13 )
        {
            is_present = false;

            do_One_Button_Alert ( kAlertStopAlert,
                                kAlertStringNoColorQuickDrawError,
                                kAlertStringNoColorQuickDrawExplanation,
                                "\pOK" );
        }
        else
        {
            is_present = true;
        }
    }
    else
    {
        do_Bail_From_Gestalt ( err );
    }

    return ( is_present );
}

//
// gestaltDisplayMgrColorSyncAware
// Version 2 appeared with System 7.5 Upgrade 2.0.
// The codification of the version number changed with version 2.
// 3.6.4 = 0x00030604
// This function allows us to use the same format as all our
// other functions.
int do_Check_For_Display_Manager ( short version )
{
    OSErr      err = noErr;
    Boolean     is_present = false;
    long        result = 0L;

    err = Gestalt ( gestaltDisplayMgrAttr, &result );
    if ( err == noErr )
    {
        if ( do_Is_Bit_Set( result, gestaltDisplayMgrPresent ) )
        {
            err = Gestalt ( gestaltDisplayMgrVers, &result );
            if ( err == noErr )
            {
                // The result is BCD, where the high bytes are the major
                // version number and the low bytes are minor version number.
                long new_version = 0L;
                new_version = ( (version & 0xF000) << 12 )
                            + ( (version & 0x0F00) << 8 )
                            + ( (version & 0x00F0) << 4 )
                            + ( (version & 0x000F) );

                if( result >= new_version )
                    is_present = true;
            }
        }
        else
        {
            do_Bail_From_Gestalt ( err );
        }

        return ( is_present );
    }
}

//
int do_Check_For_QuickTime ( short version )
{
    OSErr      err = noErr;
```



```
Boolean    is_present = false;
long       version_result = 0L;
long       features_result = 0L;

// top 3 bytes of the gestalt result are used
// hi-byte of hi-word is major revision number
// lo-byte of hi-word is minor revision number
// hi-byte of lo-word is the release stage

//short    development = 0x2000; // $20
//short    alpha      = 0x4000; // $40
//short    beta       = 0x6000; // $60
//short    final      = 0x8000; // $80
short      release    = 0x8000; // $80

err = Gestalt ( gestaltQuickTimeVersion, &version_result );
if ( err == noErr )
{
    err = Gestalt ( gestaltQuickTimeFeatures, &features_result );
    if ( err == noErr )
    {
        if ( do_Is_Bit_Set ( features_result, gestaltPPCQuickTimeLibPresent ) )
        {
            if ( HIWORD ( version_result ) < version )
            {
                is_present = false;

                do_One_Button_Alert ( kAlertStopAlert,
                                     kAlertStringNoQuickTimeError,
                                     kAlertStringPre30QuickTimeErr,
                                     "\pOK" );
            }
            else
            {
                is_present = true;

                if ( LOWORD ( version_result ) < release )
                {
                    do_One_Button_Alert ( kAlertNoteAlert,
                                         kAlertStringPreReleaseQuickTimeError,
                                         kAlertStringPreReleaseQuickTimeExplanation,
                                         "\pOK" );
                }
            }
        }
        else
        {
            is_present = false;

            do_One_Button_Alert ( kAlertStopAlert,
                                 kAlertStringNoQuickTimePowerPlugError,
                                 kAlertStringNoQuickTimePowerPlugExplanation,
                                 "\pOK" );
        }
    }
    else
    {
        do_Bail_From_Gestalt ( err );
    }
}
else
{
    do_Bail_From_Gestalt ( err );
}

return ( is_present );
}

//
int do_Check_For_Open_Transport ( short version )
{
    OSErr    err = noErr;
    Boolean   is_present = false;
    long      presence_result = 0L;
    long      version_result = 0L;

    // Note:
    // If your application uses Open Transport, it should
    // determine whether it is present using the
    // InitOpenTransport function. Do not use Gestalt for this.
    // The InitOpenTransport function performs all the right
    // checks for you.

    err = Gestalt ( gestaltOpenTpt, &presence_result );
    if ( err == noErr && presence_result != 0 )
    {
        // We first check the version.
    }
}
```

```
err = Gestalt ( gestaltOpenTptVersions, &version_result );
if ( err == noErr )
{
    //      DEBUG_VAR_PRINT("OT version is %hu", HIWORD ( version_result ));
    //      DEBUG_VAR_PRINT("OT release is %hu", LOWORD ( version_result ));

    //      If the version is too low, we don't bother with anything else and bail.
    if ( HIWORD ( version_result ) < version )
    {
        is_present = false;

        do_One_Button_Alert ( kAlertStopAlert,
                              kAlertStringNoOpenTransportError,
                              kAlertStringNoOpenTransportExplanation,
                              "\pOK" );
    }
    //      If the version is ok, we need to check features.
    else
    {
        is_present = true;

        //      We check if AppleTalk is present...
        /*      if ( !do_Is_Bit_Set ( presence_result, gestaltOpenTptAppleTalkPresentBit ) )
        {
            is_present = false;

            do_One_Button_Alert ( kAlertStopAlert,
                                  kAlertStringNoAppleTalkError,
                                  kAlertStringNoAppleTalkExplanation,
                                  "\pOK" );
        }
        */
    }
    //      If we got a gestalt error, we bail.
    else
    {
        do_Bail_From_Gestalt ( err );
    }
}
//      If there was an error with the gestalt call, we bail.
else if ( err != noErr )
{
    do_Bail_From_Gestalt ( err );
}
//      If the gestalt call went ok, but there was no response, OT is not available.
else
{
    is_present = false;

    do_One_Button_Alert ( kAlertStopAlert,
                          kAlertStringNoOpenTransportError,
                          kAlertStringNoOpenTransportExplanation,
                          "\pOK" );
}

return ( is_present );
}

//
int do_Check_For_Drag_Manager ( short version )
{
    OSErr      err = noErr;
    Boolean    is_present = false;
    long       result = 0L;

    err = Gestalt ( gestaltDragMgrAttr, &result );
    if ( err == noErr )
    {
        if ( do_Is_Bit_Set ( result, gestaltDragMgrPresent ) )
        {
            is_present = true;
        }
        else
        {
            is_present = false;

            do_One_Button_Alert ( kAlertStopAlert,
                                  kAlertStringNoDragManagerError,
                                  kAlertStringNoDragManagerExplanation,
                                  "\pOK" );
        }
    }
    else
    {
        do_Bail_From_Gestalt ( err );
    }
}
```

```
    return ( is_present );
}

//
Boolean do_Check_For_Nav_Services ( void )
{
    return( NavServicesAvailable() );
}

//
// This now works for version 1.6 and up, but not for 1.5.4
// and earlier. When QD3D was rolled into QuickTime, the version
// result was changed to match that of QuickTime, unlike its
// earlier format (which was 0x11223344, where 0x1122 was the BCD
// major version number and 0x3344 was the BCD minor version -
// i.e. 1.5.4 was 0x00010504)

int do_Check_For_QD3D ( short version )
{
    OSErr          err = noErr;
    Boolean        is_present = false;
    long           Attr_result = 0L;
    long           Version_result = 0L;

    err = Gestalt ( gestaltQD3D, &Attr_result );
    if ( err == noErr )
    {
        if ( do_Is_Bit_Set ( Attr_result, gestaltQD3DPresent ) )
        {
            is_present = true;

            err = Gestalt ( gestaltQD3DVersion, &Version_result );
            if ( err == noErr )
            {
                DEBUG_VAR_PRINT("QD3D Hi Word = %#06x", HIWORD ( Version_result ));
                DEBUG_VAR_PRINT("QD3D Lo Word = %#06x", LOWORD ( Version_result ));

                if ( HIWORD ( Version_result ) < version )
                {
                    is_present = false;

                    do_One_Button_Alert ( kAlertStopAlert,
                                          kAlertStringNoQD3DVersionError,
                                          kAlertStringNoQD3DExplanation,
                                          "\pOK" );
                }
                else
                {
                    is_present = true;
                }
            }
            else
            {
                do_Bail_From_Gestalt ( err );
            }
        }
        else
        {
            is_present = false;

            do_One_Button_Alert ( kAlertStopAlert,
                                  kAlertStringNoQD3DError,
                                  kAlertStringNoQD3DExplanation,
                                  "\pOK" );
        }
    }
    else
    {
        do_Bail_From_Gestalt ( err );
    }

    return ( is_present );
}

//
int do_Check_For_QD3D_Viewer ( short version )
{
    OSErr          err = noErr;
    Boolean        is_present = false;
    long           result = 0L;

    err = Gestalt ( gestaltQD3DViewer, &result );
    if ( err == noErr )
    {
        if ( do_Is_Bit_Set ( result, gestaltQD3DViewerPresent ) )
        {

```

```
        is_present = true;
    }
    else
    {
        is_present = false;

        do_One_Button_Alert ( kAlertStopAlert,
                              kAlertStringNoQD3DViewerError,
                              kAlertStringNoQD3DViewerExplanation,
                              "\pOK" );
    }
}
else
{
    do_Bail_From_Gestalt ( err );
}

return ( is_present );
}

//
int do_Check_For_Apple_Events ( short version )
{
    OSErr      err = noErr;
    Boolean    is_present = false;
    long       result = 0L;

    err = Gestalt ( gestaltAppleEventsAttr, &result );
    if ( err == noErr )
    {
        if ( do_Is_Bit_Set ( result, gestaltAppleEventsPresent ) )
        {
            is_present = true;
        }
        else
        {
            is_present = false;

            do_One_Button_Alert ( kAlertStopAlert,
                                  kAlertStringNoAppleEventsError,
                                  kAlertStringNoAppleEventsExplanation,
                                  "\pOK" );
        }
    }
    else
    {
        do_Bail_From_Gestalt ( err );
    }

    return ( is_present );
}

//
#pragma mark -
//
void do_Bail_From_Gestalt ( OSErr err )
{
    switch ( err )
    {
        case gestaltUnknownErr:
        {
            do_One_Button_Alert ( kAlertStopAlert, kAlertStringGestaltManagerError, kAlertString_gestaltUnknownErr,
                                  "\pOK" );
            break;
        }
        case gestaltUndefSelectorErr:
        {
            do_One_Button_Alert ( kAlertStopAlert, kAlertStringGestaltManagerError,
                                  kAlertString_gestaltUndefSelectorErr, "\pOK" );
            break;
        }
        case gestaltDupSelectorErr:
        {
            do_One_Button_Alert ( kAlertStopAlert, kAlertStringGestaltManagerError,
                                  kAlertString_gestaltDupSelectorErr, "\pOK" );
            break;
        }
        case gestaltLocationErr:
        {
            do_One_Button_Alert ( kAlertStopAlert, kAlertStringGestaltManagerError, kAlertString_gestaltLocationErr,
                                  "\pOK" );
            break;
        }
        default:
    }
```

```
    {  
        break;  
    }  
}  
ExitToShell();  
}
```

```
//  
//  
//  
#ifndef __my_macros__  
#define __my_macros__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
#if PRAGMA_ONCE  
#pragma once  
#endif  
  
#ifdef cplusplus  
extern "C" {  
#endif  
  
#ifndef MIN  
#define MIN(a,b) ( ( (a) < (b) ) ? (a) : (b) )  
#endif  
#ifndef MAX  
#define MAX(a,b) ( ( (a) > (b) ) ? (a) : (b) )  
#endif  
#ifndef ABS  
#define ABS(x) ( ( (x) < 0 ) ? ( -1 * (x) ) : (x) )  
#endif  
#ifndef CMP  
#define CMP(a,b) ( ( (a) < (b) ) ? -1 : ( ( (a) > (b) ) ? 1 : 0 ) )  
#endif  
#ifndef SGN  
#define SGN(a) ( ( (a) < 0 ) ? -1 : 1 )  
#endif  
  
// These macros return the corners of the specified rectangle as points.  
#ifndef TOPLEFT  
#define TOPLEFT(rect) (((Point *) &(rect))[0])  
#endif  
#ifndef BOTRIGHT  
#define BOTRIGHT(rect) (((Point *) &(rect))[1])  
#endif  
// !!! Need to define two new macros to get the correct bottom right point of a rect.  
// It should subtract 1 from both the right and bottom members to get the point inside the rect.  
  
// These macros return the width or height of the specified rectangle.  
#ifndef WIDTH  
#define WIDTH(rect) (((rect).right) - ((rect).left))  
#endif  
#ifndef HEIGHT  
#define HEIGHT(rect) (((rect).bottom) - ((rect).top))  
#endif  
  
// These macros return the specified part of a long or short.  
// #define HIWORD(x) ((short)((long)(x) >> 16))  
// #define LOWORD(x) ((short)(x))  
  
#ifndef HIWORD  
#define HIWORD(x) (((x) >> 16) & 0xFFFF)  
#endif  
#ifndef LOWORD  
#define LOWORD(x) ((x) & 0xFFFF)  
#endif  
#ifndef HIBYTE  
#define HIBYTE(x) ( (unsigned char) (((x) >> 8) & 0xFF) )  
#endif  
#ifndef LOBYTE  
#define LOBYTE(x) ( (unsigned char) ((x) & 0xFF) )  
#endif  
  
#ifndef fixed1  
#define fixed1 ((Fixed) 0x00010000L)  
#endif  
  
#ifndef fract1  
#define fract1 ((Fract) 0x40000000L)
```

```
#endif

// Fixed point macros
#ifndef FixedRound
#define FixedRound(a) ((short)((Fixed)(a) + fixed1 / 2 >> 16))
#endif
#ifndef FixedSquareRoot
#define FixedSquareRoot(a) ((Fixed)FractSquareRoot(a) + 64 >> 7)
#endif
#ifndef FixedTruncate
#define FixedTruncate(a) ((short)((Fixed)(a) >> 16))
#endif
#endif

#ifndef FixedToFract
#define FixedToFract(a) ((Fract)(a) << 14)
#endif
#ifndef FractToFixed
#define FractToFixed(a) ((Fixed)(a) + 8192L >> 14)
#endif
#endif

#ifndef FixedToInt
#define FixedToInt(a) ((short)((Fixed)(a) + fixed1 / 2 >> 16))
#endif
#ifndef IntToFixed
#define IntToFixed(a) ((Fixed)(a) << 16)
#endif
#endif

#ifndef FixedToFloat
#define FixedToFloat(a) ((float)(a) / fixed1)
#endif
#ifndef FloatToFixed
#define FloatToFixed(a) ((Fixed)((float)(a) * fixed1))
#endif
#endif

// u8Fixed8Number: This type represents a fixed unsigned 2 byte/16 bit
// quantity which has 8 fractional bits. An example of this encoding is:
// 0 0000h
// 1.0 0100h
// 255 + (255/256) FFFFh
#ifndef UShortFixedToFloat
#define UShortFixedToFloat(a) ((float)(a) / 0x0100)
#endif
#ifndef FloatToUShortFixed
#define FloatToUShortFixed(a) ((unsigned short)((float)(a) * 0x0100))
#endif
#endif

#ifndef FractToFloat
#define FractToFloat(a) ((float)(a) / fract1)
#endif
#ifndef FloatToFract
#define FloatToFract(a) ((Fract)((float)(a) * fract1))
#endif
#endif

#ifndef ToUpper
#define ToUpper(c) ( ((c >= 'a') && (c <= 'z')) ? c - 'a' + 'A' : c )
#endif
#endif

// Generally useful equates of basic data element sizes. One byte consists of eight
// bits (kBitsPerByte), one word consists of two bytes (kBytesPerWord), and one long word
// consists of two words (kWordsPerLWord) or four bytes (kBytesPerLWord). There are 16
// bits in a word (kBitsPerWord), and 32 bits in a long word (kBitsPerLWord).
#define kBitsPerByte 8
#define kBytesPerWord 2
#define kBitsPerWord (kBitsPerByte * kBytesPerWord)
#define kWordsPerLWord 2
#define kBytesPerLWord (kBytesPerWord * kWordsPerLWord)
#define kBitsPerLWord (kBitsPerByte * kBytesPerLWord)

// These are shorthand for the number of bytes needed for each data type.
// These are used extensively in the Photoshop routines.
#define A_CHAR (sizeof(char))
#define A_SHORT (sizeof(short))
#define A_LONG (sizeof(long))
#define A_FLOAT (sizeof(float))

// Converts the number of bits nBits to its equivalent number of rowBytes,
// making sure to account for the fact that rowBytes must be even.
#define BITS2ROWBYTES(nBits) ( ( (nBits) - 1 ) / kBitsPerWord + 1 ) * kBytesPerWord )
#define BITS2BYTES(nBits) ( ( nBits - 1 + kBitsPerByte ) / kBitsPerByte )

//
// these are the printing macros which are used for all printing
// they could be modified to work in an application as some kind of dialog box

// define the appropriate variable to turn on a bunch of debugging prints
#ifndef PRINT_DEBUG_TO_FILE
```

```
// To use the debug_to_file routines:
// 1. Include DECLARE_DEBUG_FILE_PTR as a global in main.c
// 2. Include DECLARE_EXTERN_DEBUG_FILE_PTR as a global in any additional files
// 3. Create and open the file using OPEN_DEBUG_FILE("filename","mode") early in main{}
// 4. Close the file with CLOSE_DEBUG_FILE late in main{}

#define DEBUG_PATH outFilePtr

#define DECLARE_DEBUG_FILE_PTR FILE *DEBUG_PATH=NULL;
// #define OPEN_DEBUG_FILE(text) {DEBUG_PATH=fopen(text,"w"); if(DEBUG_PATH==NULL){SysBeep(1);exit(-1);} char
time_str[32]; time_t now_t; struct tm *now_tm; now_t=time(NULL); now_tm=localtime(&now_t);
strftime(time_str,(size_t)31,"%b %d, %Y - %I:%M:%S %p",now_tm); fprintf(DEBUG_PATH,"OPEN: File opened
%s",time_str); fflush(DEBUG_PATH);}
#define OPEN_DEBUG_FILE(text,mode) \
{ \
    DEBUG_PATH=fopen(text,mode);\
    if(DEBUG_PATH==NULL)\
    { \
        SysBeep(1);\
        exit(-1);\
    } \
    else \
    { \
        char time_str[32]; \
        time_t now_t; \
        struct tm *now_tm; \
        now_t=time(NULL); \
        now_tm=localtime(&now_t); \
        strftime(time_str,(size_t)31,"%b %d, %Y - %I:%M:%S %p",now_tm); \
        fprintf(DEBUG_PATH,"OPEN: File opened %s",time_str); \
        fflush(DEBUG_PATH); \
    } \
}
#define CLOSE_DEBUG_FILE \
{ \
    char time_str[32]; \
    time_t now_t; \
    struct tm *now_tm; \
    now_t=time(NULL); \
    now_tm=localtime(&now_t); \
    strftime(time_str,(size_t)31,"%b %d, %Y - %I:%M:%S %p",now_tm); \
    fprintf(DEBUG_PATH,"\nCLOSE: File closed %s\n\n",time_str); \
    fclose(DEBUG_PATH); \
    fflush(DEBUG_PATH); \
}

#define DECLARE_EXTERN_DEBUG_FILE_PTR extern FILE *DEBUG_PATH;

#else

#define DECLARE_DEBUG_FILE_PTR
#define OPEN_DEBUG_FILE(text,mode)
#define CLOSE_DEBUG_FILE

#define DECLARE_EXTERN_DEBUG_FILE_PTR

#ifdef PRINT_DEBUG_TO_STDIO

#define DEBUG_PATH stdout

#else

#undef PRINTF_MESSAGES
#undef PRINT_STATUS_MESSAGES
#undef PRINT_DEBUG_MESSAGES
#undef PRINT_WARNING_MESSAGES
#undef PRINT_ERROR_MESSAGES

#endif

#endif

#ifdef PRINTF_MESSAGES

#define PRINTF(text) {fprintf(DEBUG_PATH,"\n"); fprintf(DEBUG_PATH,text); fflush(DEBUG_PATH);}
#define EXTRA_PRINTF(text) {fprintf(DEBUG_PATH,text); fflush(DEBUG_PATH);}
#define VAR_PRINTF(text,variable) {fprintf(DEBUG_PATH,"%s",variable); fprintf(DEBUG_PATH,text ## , ## variable); fflush(DEBUG_PATH);}
#define EXTRA_VAR_PRINTF(text,variable) {fprintf(DEBUG_PATH,text ## , ## variable); fflush(DEBUG_PATH);}

#else

#define PRINTF(text) {}
#define EXTRA_PRINTF(text) {}
#define VAR_PRINTF(text,variable) {}

#endif
```



```
#define EXTRA_VAR_PRINTF(text,variable) {}

#endif

#ifdef PRINT_STATUS_MESSAGES

#define STATUS_PRINT(text) {fprintf(DEBUG_PATH, "\nSTATUS: "); fprintf(DEBUG_PATH, text); fflush(DEBUG_PATH);}
#define STATUS_EXTRA_PRINT(text) {fprintf(DEBUG_PATH, text); fflush(DEBUG_PATH);}
#define STATUS_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, "\nSTATUS: "); fprintf(DEBUG_PATH, text ## , ##
variable); fflush(DEBUG_PATH);}
#define STATUS_EXTRA_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, text ## , ## variable); fflush(DEBUG_PATH);}
#define STATUS_TIME_PRINT (char time_str[16]; time_t now_t; struct tm *now_tm; now_t=time(NULL);
now_tm=localtime(&now_t); strftime(time_str, (size_t)15, "%I:%M:%S %p", now_tm); fprintf(DEBUG_PATH, "\nTIME:
%s", time_str); fflush(DEBUG_PATH);}
#define STATUS_TIME_DATE_PRINT (char time_str[32]; time_t now_t; struct tm *now_tm; now_t=time(NULL);
now_tm=localtime(&now_t); strftime(time_str, (size_t)31, "%b %d, %Y - %I:%M:%S %p", now_tm);
fprintf(DEBUG_PATH, "\nTIME: %s", time_str); fflush(DEBUG_PATH);}

#else

#define STATUS_PRINT(text) {}
#define STATUS_EXTRA_PRINT(text) {}
#define STATUS_VAR_PRINT(text,variable) {}
#define STATUS_EXTRA_VAR_PRINT(text,variable) {}
#define STATUS_TIME_PRINT {}
#define STATUS_TIME_DATE_PRINT {}

#endif

#ifdef PRINT_DEBUG_MESSAGES

#define DEBUG_PRINT(text) {fprintf(DEBUG_PATH, "\nDEBUG: "); fprintf(DEBUG_PATH, text); fflush(DEBUG_PATH);}
#define DEBUG_EXTRA_PRINT(text) {fprintf(DEBUG_PATH, text); fflush(DEBUG_PATH);}
#define DEBUG_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, "\nDEBUG: "); fprintf(DEBUG_PATH, text ## , ##
variable); fflush(DEBUG_PATH);}
#define DEBUG_EXTRA_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, text ## , ## variable); fflush(DEBUG_PATH);}

#else

#define DEBUG_PRINT(text) {}
#define DEBUG_EXTRA_PRINT(text) {}
#define DEBUG_VAR_PRINT(text,variable) {}
#define DEBUG_EXTRA_VAR_PRINT(text,variable) {}

#endif

#ifdef PRINT_WARNING_MESSAGES

#define WARNING_PRINT(text) {fprintf(DEBUG_PATH, "\nWARNING: "); fprintf(DEBUG_PATH, text); fflush(DEBUG_PATH);}
#define WARNING_EXTRA_PRINT(text) {fprintf(DEBUG_PATH, text); fflush(DEBUG_PATH);}
#define WARNING_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, "\nWARNING: "); fprintf(DEBUG_PATH, text ## , ##
variable); fflush(DEBUG_PATH);}
#define WARNING_EXTRA_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, text ## , ## variable); fflush(DEBUG_PATH);}

#else

#define WARNING_PRINT(text) {}
#define WARNING_EXTRA_PRINT(text) {}
#define WARNING_VAR_PRINT(text,variable) {}
#define WARNING_EXTRA_VAR_PRINT(text,variable) {}

#endif

#ifdef PRINT_ERROR_MESSAGES

// #define ERROR_PRINT(text) {fprintf(DEBUG_PATH, "\nERROR: "); fprintf(DEBUG_PATH, text);
fprintf(DEBUG_PATH, "\n\nExiting...\n"); fflush(DEBUG_PATH); exit(-1);}
#define ERROR_PRINT(text) {fprintf(DEBUG_PATH, "\nERROR: "); fprintf(DEBUG_PATH, text);
fprintf(DEBUG_PATH, "\n\nExiting...\n"); CLOSE_DEBUG_FILE; exit(-1);}
// #define ERROR_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, "\nERROR: "); fprintf(DEBUG_PATH, text ## , ##
variable); fprintf(DEBUG_PATH, "\n\nExiting...\n"); fflush(DEBUG_PATH); exit(-1);}
#define ERROR_VAR_PRINT(text,variable) {fprintf(DEBUG_PATH, "\nERROR: "); fprintf(DEBUG_PATH, text ## , ##
variable); fprintf(DEBUG_PATH, "\n\nExiting...\n"); CLOSE_DEBUG_FILE; exit(-1);}

#else

#define ERROR_PRINT(text) {}
#define ERROR_VAR_PRINT(text,variable) {}

#endif

//
```

```
#ifdef __cplusplus
}
#endif

#endif /* __my_macros__ */
```

```
//
//                                     ©1998-2000 bergdesign inc.
//
#ifdef __my_menus__
#define __my_menus__

#ifdef ACCESSOR_CALLS_ARE_FUNCTIONS
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1
#endif
#ifdef OPAQUE_TOOLBOX_STRUCTS
#define OPAQUE_TOOLBOX_STRUCTS 1
#endif

#ifdef __APPLE_CC__
#include <Carbon/Carbon.h>
#else
#ifdef TARGET_API_MAC_CARBON
#include <Carbon.h>
#else
#include <Menus.h>
#include <Devices.h>
#endif
#endif

#include "my_alerts.h"
#include "my_gestalts.h"

#ifdef __cplusplus
extern "C" {
#endif

// MENU resource IDs
// Note that some menu functions take resource IDs,
// while others take menu IDs. You should set the
// menu IDs to be the same as the resource IDs
// to eliminate confusion.
enum
{
    kMenuBarID                = 128,

    kAppleMenuID              = 128,
    kAboutBoxItem              = 1,

    kFileMenuID                = 129,

    kEditMenuID                = 130
};

void do_Init_Menubar ( void );
long do_Make_Menu_Result ( short, short );
void do_Show_Menu_Bar ( Boolean );

#ifdef TARGET_API_MAC_OS8 && !TARGET_API_MAC_CARBON
void do_Open_Desk_Accessory ( short );
#endif

#ifdef __cplusplus
}
#endif

#endif /* __my_menus__ */
```



```
//  
void do_Show_Menu_Bar ( Boolean show )  
{  
    OSErr          err = noErr;  
    //GDHandle      main_screen;  
    //Rect          main_screen_rect;  
    //RgnHandle     main_screen_rgn = NULL;  
    //short         mbar_height = 0;  
    //static RgnHandle mbar_rgn = NULL;  
    //static short   old_mbar_height = 0;  
  
    // If we're running under 8.5, we need to use the new ShowMenuBar() and  
    // HideMenuBar() functions. We are advised to avoid writing to the GrayRgn  
    // at all costs. One of the problems is that the gray region includes the  
    // menu bar area under 8.5 and later, but does not include it under pre 8.5  
    // systems. This sucks.  
  
    if ( do_Check_For_System_Version( 0x0850 ) ) // 8.5 or later  
    {  
        DEBUG_PRINT("Using 8.5+ show/hide menu bar routines");  
  
        if ( show ) // show menu bar  
        {  
            if ( !IsMenuBarVisible() )  
            {  
                ShowMenuBar();  
                InvalMenuBar();  
            }  
        }  
        else // hide menu bar  
        {  
            if ( IsMenuBarVisible() )  
            {  
                HideMenuBar();  
            }  
        }  
    }  
    else // pre 8.5  
    {  
        /*  
        DEBUG_PRINT("Using pre 8.5 show/hide menu bar routines");  
  
        if ( show ) // show menu bar  
        {  
            mbar_height = GetMBarHeight();  
  
            // If the menu bar height is not 0, the stored height is 0, or we don't  
            // have a saved menu bar region, it means the menu bar is already visible.  
            if ( mbar_height != 0 || old_mbar_height == 0 || mbar_rgn == NULL )  
                return;  
  
            // We restore the menu bar height to the saved value.  
            LMSetMBarHeight ( old_mbar_height );  
            old_mbar_height = 0;  
  
            // Now we put the gray region back like it was. We take the stored,  
            // menu bar region and subtract it from the enlarged gray region  
            // that includes the menu bar area.  
            DiffRgn ( GetGrayRgn(), mbar_rgn, GetGrayRgn() );  
  
            // Finally, draw the menu bar.  
            DrawMenuBar();  
  
            if ( mbar_rgn != NULL )  
            {  
                DisposeRgn ( mbar_rgn );  
                mbar_rgn = NULL;  
            }  
        }  
        else // hide menu bar  
        {  
            mbar_height = GetMBarHeight();  
  
            // If the menu bar height is 0, the stored height is not 0, or we have  
            // a saved menu bar region, it means we've already hidden the menu bar.  
            if ( mbar_height == 0 || old_mbar_height != 0 || mbar_rgn != NULL )  
                return;  
  
            // We store the current menu bar height and hide the menu bar.  
            old_mbar_height = mbar_height;  
            LMSetMBarHeight(0);  
  
            // We create a temporary region of the entire main screen area.  
            main_screen_rgn = NewRgn();  
            main_screen = GetMainDevice();  
            main_screen_rect = (*main_screen)->gdRect;  
            RectRgn ( main_screen_rgn, &main_screen_rect );
```

```
// We create a new region to store the menu bar area.
// We've already checked above for an empty region handle.
mbar_rgn = NewRgn();

DiffRgn ( main_screen_rgn, GetGrayRgn(), mbar_rgn );

UnionRgn ( GetGrayRgn(), mbar_rgn, GetGrayRgn() );

// The NULL in the window record (WindowPeek) parameter causes the desktop to be updated.
PaintOne ( NULL, mbar_rgn );

if ( main_screen_rgn )
    DisposeRgn ( main_screen_rgn );
}
*/
}
}

#ifdef TARGET_API_MAC_OS8 && !TARGET_API_MAC_CARBON

//
// _____ Open a pre-Carbon desk accessory

void do_Open_Desk_Accessory ( short menu_item )
{
    Str255      menu_name;
    CGrafPtr    port;
    GDHandle     gdh;

    GetGWorld ( &port, &gdh );

    GetMenuItemText ( GetMenu(kAppleMenuID ), menu_item, menu_name );
    OpenDeskAcc ( menu_name );

    SetGWorld ( port, gdh );
}

#endif
```

```
//  
//  
//
```

```
#ifndef __my_quickdraw__  
#define __my_quickdraw__
```

```
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif
```

```
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Quickdraw.h>  
#include <QDOffscreen.h>  
#include <TextUtils.h>  
#include <MacWindows.h>  
#include <ColorPicker.h>  
#endif  
#endif
```

```
#include <math.h>
```

```
#include "my_alerts.h"  
#include "my_macros.h"  
#include "my_windows.h"  
#include "my_colors.h"
```

```
#ifdef __cplusplus  
extern "C" {  
#endif
```

```
typedef struct char_pixel  
{  
    unsigned char alpha;  
    unsigned char red;  
    unsigned char green;  
    unsigned char blue;  
} RGBCharColor;
```

```
typedef struct short_pixel  
{  
    unsigned short alpha;  
    unsigned short red;  
    unsigned short green;  
    unsigned short blue;  
} RGBShortColor;
```

```
typedef struct float_pixel  
{  
    float alpha;  
    float red;  
    float green;  
    float blue;  
} RGBFloatColor;
```

```
struct ColorPenState  
{  
    Boolean        colorPort;  
    RGBColor       foreColor;  
    RGBColor       backColor;  
    PenState       pen;  
    SInt16         textFont;  
    StyleField     textFace;  
    SInt16         textMode;  
    SInt16         textSize;  
    PixPatHandle   pnPixPat;  
    PixPatHandle   bkPixPat;  
    Pattern        bkPat;  
    UInt32         fgColor;  
    UInt32         bkColor;  
};
```

```
typedef struct ColorPenState ColorPenState;
```

```
enum
```

```
{
    PlatinumScrollBorderActive      = 1,
    PlatinumScrollBorderInactive    = 2
};

// #define IsColorGrafPort( port ) (((port)->portBits.rowBytes & 0xC000) == 0xC000)
#define GetCurrentDepth( port ) ( IsColorGrafPort( port ) ? (*(CGrafPtr)port)->portPixMap.pixelSize : 1 )

RGBFloatColor    RGBCharColor_to_RGBFloatColor ( RGBCharColor );
RGBCharColor     RGBFloatColor_to_RGBCharColor ( RGBFloatColor );
RGBFloatColor    RGBShortColor_to_RGBFloatColor ( RGBShortColor );
RGBShortColor    RGBFloatColor_to_RGBShortColor ( RGBFloatColor );
RGBFloatColor    RGBColor_to_RGBFloatColor ( RGBColor );
RGBColor         RGBFloatColor_to_RGBColor ( RGBFloatColor );
RGBCharColor     RGBColor_to_RGBCharColor ( RGBColor );
RGBColor         RGBCharColor_to_RGBColor ( RGBCharColor );

RGBFloatColor    do_Clip_RGBFloatColor ( RGBFloatColor my_float_color );

Boolean          do_Get_Color_Picker_Color ( RGBColor *, const unsigned char * );
RGBColor         do_Make_RGBColor ( unsigned short, unsigned short, unsigned short );

RGBFloatColor    gamma_to_linear_rgb ( RGBFloatColor, float );
RGBFloatColor    linear_to_gamma_rgb ( RGBFloatColor, float );

void             do_Draw_Gradation( Rect, RGBColor, RGBColor, int );

OSErr            do_Get_New_GWorld ( GWorldPtr *, int, const Rect *, Boolean );
//GWorldPtr      do_Get_New_GWorld ( const Rect *, Sint8, Boolean );
void             do_Erase_GWorld ( GWorldPtr );

int              do_Check_For_QDError ( void );
int              do_Check_For_MemError ( void );

pascal void      do_Global_to_Local_Rect ( Rect * );
pascal void      do_Local_to_Global_Rect ( Rect * );
pascal Rect      do_Make_Rect ( int, int, int, int );
Rect             do_Max_Inscribed_Square( Rect );
Rect             do_Max_Inscribed_Rect( Rect, Rect );
Point            do_Midpoint( Point, Point );

void             do_Draw_Styled_Text( short, Rect * );
void             do_Draw_Styled_Transparent_Text (short, Rect *, short );

//void           GetColorAndPenState ( ColorPenState * );
//void           SetColorAndPenState ( ColorPenState * );
void             NormalizeColorAndPen ( void );
void             InverseNormalizeColorAndPen ( void );
void             GrayColorAndPen ( void );

void             do_Track_Cursor_With_Square ( Point, short, void (*)(void) );
void             do_Track_Cursor_With_Circle ( Point, short );

// Window color table accessors for appearance manager uses

void             do_Set_Pen ( short );
void             do_Set_Pen_To_WCTB_Color ( short );
Boolean          do_Get_Window_RGBColor ( WindowPtr, short, RGBColor * );
CTabHandle       do_Get_Window_Color_Table ( WindowPtr );
Boolean          do_Get_RGBColor_From_Color_Table ( CTabHandle, short, RGBColor * );

// Functions to move the cursor

void             do_Move_Mouse( Point pt );
void             do_Mouse_Couple( void );
void             do_Mouse_Decouple( void );

#ifdef __cplusplus
}
#endif

#endif /* __my_quickdraw__ */
```


~/Stuff/Code/Common Code/C Files/my_quickdraw.c
 Saved: Tuesday, December 11, 2001 6:20:21 PM

```
// _____ ©1998-2001 bergdesign inc.
//
//
#include "my_quickdraw.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

// Static globals -- low memory mouse globals
// Should use LowMem.h, but direct access is faster than a function call.
Point *gTempMouse = (Point*) 0x0828;
Point *gRawMouse = (Point*) 0x082C;
Point *gMouseLoc = (Point*) 0x0830;
char *gNewMouse = (char*) 0x08CE;
char *gCouple = (char*) 0x08CF;

// _____ RGBCharColor to RGBFloatColor
//
RGBFloatColor RGBCharColor_to_RGBFloatColor ( RGBCharColor my_char_color )
{
  RGBFloatColor my_float_color;

  my_float_color.alpha = (float)(my_char_color.alpha)/255.0;
  my_float_color.red = (float)(my_char_color.red)/255.0;
  my_float_color.green = (float)(my_char_color.green)/255.0;
  my_float_color.blue = (float)(my_char_color.blue)/255.0;

  return my_float_color;
}

// _____ RGBFloatColor to RGBCharColor
//
RGBCharColor RGBFloatColor_to_RGBCharColor ( RGBFloatColor my_float_color )
{
  RGBCharColor my_char_color;

  my_char_color.alpha = (unsigned char)( my_float_color.alpha * 255.0 );
  my_char_color.red = (unsigned char)( my_float_color.red * 255.0 );
  my_char_color.green = (unsigned char)( my_float_color.green * 255.0 );
  my_char_color.blue = (unsigned char)( my_float_color.blue * 255.0 );

  return my_char_color;
}

// _____ RGBShortColor to RGBFloatColor
//
RGBFloatColor RGBShortColor_to_RGBFloatColor ( RGBShortColor my_short_color )
{
  RGBFloatColor my_float_color;

  my_float_color.alpha = (float)( my_short_color.alpha ) / 65535.0;
  my_float_color.red = (float)( my_short_color.red ) / 65535.0;
  my_float_color.green = (float)( my_short_color.green ) / 65535.0;
  my_float_color.blue = (float)( my_short_color.blue ) / 65535.0;

  return my_float_color;
}

// _____ RGBFloatColor to RGBShortColor
//
RGBShortColor RGBFloatColor_to_RGBShortColor ( RGBFloatColor my_float_color )
{
  RGBShortColor my_short_color;

  my_short_color.alpha = (unsigned short)( my_float_color.alpha * 65535.0 );
  my_short_color.red = (unsigned short)( my_float_color.red * 65535.0 );
  my_short_color.green = (unsigned short)( my_float_color.green * 65535.0 );
  my_short_color.blue = (unsigned short)( my_float_color.blue * 65535.0 );

  return my_short_color;
}

// _____ RGBColor to RGBFloatColor
//
RGBFloatColor RGBColor_to_RGBFloatColor ( RGBColor my_rgb_color )
{
  RGBFloatColor my_float_color;

  my_float_color.alpha = 0.0;
  my_float_color.red = (float)( my_rgb_color.red ) / 65535.0;
  my_float_color.green = (float)( my_rgb_color.green ) / 65535.0;
  my_float_color.blue = (float)( my_rgb_color.blue ) / 65535.0;
}
```

```
    return my_float_color;
}

// _____
// _____ RGBFloatColor to RGBColor

RGBColor RGBFloatColor_to_RGBColor ( RGBFloatColor my_float_color )
{
    RGBColor      my_rgb_color;

    my_rgb_color.red    = (unsigned short)( my_float_color.red * 65535.0 );
    my_rgb_color.green  = (unsigned short)( my_float_color.green * 65535.0 );
    my_rgb_color.blue   = (unsigned short)( my_float_color.blue * 65535.0 );

    return my_rgb_color;
}

// _____
// _____ RGBColor to RGBCharColor

RGBCharColor RGBColor_to_RGBCharColor ( RGBColor my_rgb_color )
{
    RGBCharColor  my_char_color;

    my_char_color.alpha = 0;
    my_char_color.red   = (unsigned char)( my_rgb_color.red / 257 );
    my_char_color.green = (unsigned char)( my_rgb_color.green / 257 );
    my_char_color.blue  = (unsigned char)( my_rgb_color.blue / 257 );

    return my_char_color;
}

// _____
// _____ RGBCharColor to RGBColor

RGBColor RGBCharColor_to_RGBColor ( RGBCharColor my_char_color )
{
    RGBColor      my_rgb_color;

    my_rgb_color.red    = (unsigned short)( my_char_color.red * 257 );
    my_rgb_color.green  = (unsigned short)( my_char_color.green * 257 );
    my_rgb_color.blue   = (unsigned short)( my_char_color.blue * 257 );

    return my_rgb_color;
}

// _____
// _____ clip floating point values

RGBFloatColor do_Clip_RGBFloatColor ( RGBFloatColor my_float_color )
{
    if (my_float_color.red < 0.0)
        my_float_color.red = 0.0;

    if (my_float_color.red > 1.0)
        my_float_color.red = 1.0;

    if (my_float_color.green < 0.0)
        my_float_color.green = 0.0;

    if (my_float_color.green > 1.0)
        my_float_color.green = 1.0;

    if (my_float_color.blue < 0.0)
        my_float_color.blue = 0.0;

    if (my_float_color.blue > 1.0)
        my_float_color.blue = 1.0;

    if (my_float_color.alpha < 0.0)
        my_float_color.alpha = 0.0;

    if (my_float_color.alpha > 1.0)
        my_float_color.alpha = 1.0;

    return my_float_color;
}

// _____

#pragma mark -

// _____

Boolean do_Get_Color_Picker_Color ( RGBColor *the_color, const unsigned char *message )
{
    RGBColor      in_color, out_color;
```

```
Point          dialog_loc;
CGrafPtr       port;
GDHandle       gdh;
Boolean        got_color = false;

    GetGWorld ( &port, &gdh );

//  this makes the dialog appear in the center of the screen
    dialog_loc.v = dialog_loc.h = 0;

//  set the default color picker color to the stored value
    in_color = *the_color;

    DeactivateFloatersAndFirstDocumentWindow();

//  get the color picker choice, unless the user cancels
    if ( GetColor ( dialog_loc, message, &in_color, &out_color ) )
    {
        *the_color = out_color;
        got_color = true;
    }

    ActivateFloatersAndFirstDocumentWindow();

    SetGWorld ( port, gdh );

    return ( got_color );
}

//
RGBColor do_Make_RGBColor (unsigned short red, unsigned short green, unsigned short blue )
{
    RGBColor the_color;

    the_color.red   = red;
    the_color.green = green;
    the_color.blue  = blue;

    return ( the_color );
}

//
//  remove gamma from floating point pixel
RGBFloatColor gamma_to_linear_rgb ( RGBFloatColor my_float_color, float gamma )
{
    my_float_color.red   = pow (my_float_color.red, gamma);
    my_float_color.green = pow (my_float_color.green, gamma);
    my_float_color.blue  = pow (my_float_color.blue, gamma);
    my_float_color.alpha = pow (my_float_color.alpha, gamma);

    return my_float_color;
}

//
//  gamma curve floating point pixel
RGBFloatColor linear_to_gamma_rgb ( RGBFloatColor my_float_color, float gamma )
{
    float inv_gamma = 1.0/gamma;

    my_float_color.red   = pow (my_float_color.red, inv_gamma);
    my_float_color.green = pow (my_float_color.green, inv_gamma);
    my_float_color.blue  = pow (my_float_color.blue, inv_gamma);
    my_float_color.alpha = pow (my_float_color.alpha, inv_gamma);

    return my_float_color;
}

//
void do_Draw_Gradation( Rect the_rect, RGBColor start_color, RGBColor end_color, int num_steps )
{
    // Save the clip region
    RgnHandle saved_clip_rgn = NewRgn();
    GetClip ( saved_clip_rgn );

    ClipRect( &the_rect );

    num_steps = MAX( 2, num_steps );

    int height = the_rect.bottom - the_rect.top;
    int width  = the_rect.right - the_rect.left;

    float patch_width, patch_height;

    if( width >= height )
```

```
{
    patch_width = (float)width / num_steps;
    patch_height = height;
}
else
{
    patch_width = width;
    patch_height = (float)height / num_steps;
}

DEBUG_VAR_PRINT("patch width = %f",patch_width);
DEBUG_VAR_PRINT("patch height = %f",patch_height);

Rect start_rect = do_Make_Rect( the_rect.left, the_rect.top, the_rect.left + patch_width + 1, the_rect.top +
patch_height + 1 );
Rect patch_rect = start_rect;

float red_step = ( end_color.red - start_color.red ) / (float)( num_steps - 1 );
float green_step = ( end_color.green - start_color.green ) / (float)( num_steps - 1 );
float blue_step = ( end_color.blue - start_color.blue ) / (float)( num_steps - 1 );

DEBUG_VAR_PRINT("step rgb = %d",red_step);
DEBUG_EXTRA_VAR_PRINT(", %d",green_step);
DEBUG_EXTRA_VAR_PRINT(", %d",blue_step);

RGBColor the_color;

for( int i = 0; i < num_steps; i++ )
{
    the_color.red = start_color.red + ( i * red_step );
    the_color.green = start_color.green + ( i * green_step );
    the_color.blue = start_color.blue + ( i * blue_step );

    DEBUG_VAR_PRINT("step %d = ",i);
    DEBUG_EXTRA_VAR_PRINT("%d",the_color.red);
    DEBUG_EXTRA_VAR_PRINT(", %d",the_color.green);
    DEBUG_EXTRA_VAR_PRINT(", %d",the_color.blue);

    if( width >= height )
        OffsetRect( &patch_rect, i * patch_width, 0 );
    else
        OffsetRect( &patch_rect, 0, i * patch_height );

    // Fill the patch
    RGBForeColor( &the_color );
    PaintRect( &patch_rect );

    patch_rect = start_rect;
}

// Restore the old clip region
SetClip( saved_clip_rgn );
DisposeRgn( saved_clip_rgn );
}

//
#pragma mark -

//
// make new gworld

OSErr do_Get_New_GWorld ( GWorldPtr *off_gworld, int depth, const Rect *bounds, Boolean erase )
{
    PixMapHandle    pixmap_hndl;
    QDErr          err = noErr;

    if ( *off_gworld == NULL )
    {
        if ( depth == 24 || depth == 32 )
        {
            err = NewGWorld ( off_gworld, 32, bounds, NULL, NULL, keepLocal );
            if ( err == noErr )
            {
                err = do_Check_For_QDError();
                if ( err == noErr )
                {
                    DEBUG_VAR_PRINT("NewGWorld rect = %d",bounds->left);
                    DEBUG_EXTRA_VAR_PRINT(", %d",bounds->top);
                    DEBUG_EXTRA_VAR_PRINT(", %d",bounds->right);
                    DEBUG_EXTRA_VAR_PRINT(", %d",bounds->bottom);

                    if ( *off_gworld != NULL )
                    {
                        if ( depth == 24 )
                        {
                            // Don't need to change the default component values
                        }
                    }
                }
            }
        }
    }
}
```

```
        else if ( depth == 32 )
        {
            // Get a handle to the pixmap of the new gworld
            pixmap_hndl = GetGWorldPixMap ( *off_gworld );

            // Set the pixel's color depth to 4 components (channels) per pixel,
            // and 8 bits per color (channel) so that a saved PICT file contains
            // a proper alpha channel
            ( **pixmap_hndl ).cmpCount = 4;
            ( **pixmap_hndl ).cmpSize = 8;
        }

        if ( erase )
        {
            do_Erase_GWorld ( *off_gworld );
        }
        else
        {
            err = memPCErr;
        }
    }
}
}
else
{
    err = paramErr;
}
}
else
{
    err = paramErr;
}

if ( err != noErr )
{
    if ( *off_gworld != NULL )
        DisposeGWorld ( *off_gworld );
}

return ( err );
}

// _____ erase gworld

void do_Erase_GWorld ( GWorldPtr off_gworld )
{
    CGrafPtr      port;
    GDHandle      gdh;
    PixmapHandle   pixmap_hndl;

    DEBUG_PRINT("Entered do_Erase_GWorld()");

    if ( !off_gworld )
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pCan't erase the specified GWorld. The pointer is uninitialized.",
        NULL, "\pOK" );
        ExitToShell ();
    }

    GetGWorld ( &port, &gdh );

    SetGWorld ( off_gworld, NULL );

    pixmap_hndl = GetGWorldPixMap ( off_gworld );

    LockPixels ( pixmap_hndl );
    EraseRect ( &( (**pixmap_hndl).bounds ) );
    UnlockPixels ( pixmap_hndl );

    SetGWorld ( port, gdh );

    DEBUG_PRINT("Left do_Erase_GWorld()");
}

// _____

#pragma mark -

// _____

int do_Check_For_QDError ( void )
{
    QDErr      err;
    char      error_num[256];

    err = QDError();
```

```
switch ( err )
{
    case noErr: // 0
    {
        //      No error
        break;
    }
    case paramErr: // -50
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pIllegal parameter to NewGWorld().", "\pQDError", "\pOK" );
        break;
    }
    case -143:
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pCopyBits couldn't allocate required temporary memory.",
"\pQDError", "\pOK" );
        break;
    }
    case -144:
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pRan out of stack space while drawing polygon.", "\pQDError",
"\pOK" );
        break;
    }
    case noMemForPictPlaybackErr: // -145
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pInsufficient memory for drawing the picture.", "\pQDError",
"\pOK" );
        break;
    }
    case rgnTooBigError: // -147
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pRegion too big or complex.", "\pQDError", "\pOK" );
        break;
    }
    case pixMapTooDeepErr: // -148
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pPixel map is deeper than 1 bit per pixel.", "\pQDError", "\pO
);
        break;
    }
    case nsStackErr: // -149
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pInsufficient stack.", "\pQDError", "\pOK" );
        break;
    }
    case cMatchErr: // -150
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pColor2Index failed to find an index.", "\pQDError", "\pOK" );
        break;
    }
    case cTempMemErr: // -151
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pFailed to allocate memory for temporary structures.",
"\pQDError", "\pOK" );
        break;
    }
    case cNoMemErr: // -152
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pFailed to allocate memory for structure.", "\pQDError", "\pOK
);
        break;
    }
    case cRangeErr: // -153
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pRange error on color table request.", "\pQDError", "\pOK" );
        break;
    }
    case cProtectErr: // -154
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pColorTable record entry protection violation.", "\pQDError",
"\pOK" );
        break;
    }
    case cDevErr: // -155
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pInvalid type of graphics device.", "\pQDError", "\pOK" );
        break;
    }
    case cResErr: // -156
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pInvalid resolution for MakeITable.", "\pQDError", "\pOK" );
        break;
    }
    case cDepthErr: // -157
    {
        do_One_Button_Alert ( kAlertStopAlert, "\pInvalid pixel depth specified to NewGWorld.", "\pQDError",
```

```
"\pOK" );
    break;
}
case rgnTooBigErr: // -500
{
    do_One_Button_Alert ( kAlertStopAlert, "\pBitmap would convert to a region greater than 64 KB.",
"\pQDError", "\pOK" );
    break;
}

// QDError also returns MemError error codes
case memROZErr: // -99
{
    do_One_Button_Alert ( kAlertStopAlert, "\pOperation on a read-only zone.", "\pMemError returned by
QDError", "\pOK" );
    break;
}
case memFullErr: // -108
{
    do_One_Button_Alert ( kAlertStopAlert, "\pNot enough memory.", "\pMemError returned by QDError", "\pOK"
);
    break;
}
case nilHandleErr: // -109
{
    do_One_Button_Alert ( kAlertStopAlert, "\pNIL master pointer.", "\pMemError returned by QDError", "\pOK
);
    break;
}
case memWZErr: // -111
{
    do_One_Button_Alert ( kAlertStopAlert, "\pAttempt to operate on a free block.", "\pMemError returned by
QDError", "\pOK" );
    break;
}
case memPurErr: // -112
{
    do_One_Button_Alert ( kAlertStopAlert, "\pAttempt to purge a locked block.", "\pMemError returned by
QDError", "\pOK" );
    break;
}
case memBCErr: // -115
{
    do_One_Button_Alert ( kAlertStopAlert, "\pBlock check failed.", "\pMemError returned by QDError", "\pOK
);
    break;
}
case memLockedErr: // -117
{
    do_One_Button_Alert ( kAlertStopAlert, "\pBlock is locked.", "\pMemError returned by QDError", "\pOK" )
    break;
}
}

// This is the catch-all for codes that QDError returns, but that we don't know about
default:
{
    sprintf ( error_num, "Unspecified error %d.", err );
    do_c2p_str ( error_num );
    do_One_Button_Alert ( kAlertStopAlert, (StringPtr)error_num, "\pQDError", "\pOK" );
    break;
}
}

return err;
}

//
int do_Check_For_MemError ( void )
{
OSErr      err;
char        error_num[256];

    err = MemError();

    switch ( err )
    {
        case noErr: // 0
        {
            // No error
            break;
        }
        case paramErr: // -50
        {
            do_One_Button_Alert ( kAlertStopAlert, "\pError in parameter list.", "\pMemError", "\pOK" );
            break;
        }
    }
}
```

```
}
case memROZErr: // -99
{
    do_One_Button_Alert ( kAlertStopAlert, "\pOperation on a read-only zone.", "\pMemError", "\pOK" );
    break;
}
case memFullErr: // -108
{
    do_One_Button_Alert ( kAlertStopAlert, "\pNot enough memory.", "\pMemError", "\pOK" );
    break;
}
case nilHandleErr: // -109
{
    do_One_Button_Alert ( kAlertStopAlert, "\pNIL master pointer.", "\pMemError", "\pOK" );
    break;
}
case memWZErr: // -111
{
    do_One_Button_Alert ( kAlertStopAlert, "\pAttempt to operate on a free block.", "\pMemError", "\pOK" );
    break;
}
case memPurErr: // -112
{
    do_One_Button_Alert ( kAlertStopAlert, "\pAttempt to purge a locked block.", "\pMemError", "\pOK" );
    break;
}
case memBCErr: // -115
{
    do_One_Button_Alert ( kAlertStopAlert, "\pBlock check failed.", "\pMemError", "\pOK" );
    break;
}
case memLockedErr: // -117
{
    do_One_Button_Alert ( kAlertStopAlert, "\pBlock is locked.", "\pMemError", "\pOK" );
    break;
}
default:
{
    sprintf ( error_num, "Unspecified error %d.", err );
    do_c2p_str ( error_num );
    do_One_Button_Alert ( kAlertStopAlert, (StringPtr)error_num, "\pMemError", "\pOK" );
    break;
}
}

return err;
}

//
#pragma mark -

//
//
global to local rect

pascal void
do_Global_to_Local_Rect ( Rect *rect )
{
    Point    topLeft, botRight;

    topLeft.v = rect->top;
    topLeft.h = rect->left;
    botRight.v = rect->bottom;
    botRight.h = rect->right;

    GlobalToLocal ( &topLeft );
    GlobalToLocal ( &botRight );

    rect->left = topLeft.h;
    rect->top = topLeft.v;
    rect->right = botRight.h;
    rect->bottom = botRight.v;
}

//
//
local to global rect

pascal void
do_Local_to_Global_Rect ( Rect *rect )
{
    Point    topLeft, botRight;

    topLeft.v = rect->top;
    topLeft.h = rect->left;
    botRight.v = rect->bottom;
    botRight.h = rect->right;

    LocalToGlobal ( &topLeft );
```



```
LocalToGlobal ( &botRight );

rect->left = topLeft.h;
rect->top = topLeft.v;
rect->right = botRight.h;
rect->bottom = botRight.v;
}

//-----
// make rect

pascal Rect do_Make_Rect ( int left, int top, int right, int bottom )
{
    Rect    new_rect;

    new_rect.left = left;
    new_rect.top = top;
    new_rect.right = right;
    new_rect.bottom = bottom;

    return new_rect;
}

//-----
// Returns the largest square that can be inscribed in the supplied rect,
// centered in the supplied rect's coordinates.

Rect do_Max_Inscribed_Square( Rect outer )
{
    Rect inner = {0,0,0,0};
    int width = outer.right - outer.left;
    int height = outer.bottom - outer.top;

    if( width == height )
    {
        inner = outer;
    }
    else if( width > height )
    {
        inner.left    = outer.left + ( ( width - height ) / 2 );
        inner.right   = inner.left + height;
        inner.top     = outer.top;
        inner.bottom  = outer.bottom;
    }
    else // width < height
    {
        inner.left    = outer.left;
        inner.right   = outer.right;
        inner.top     = outer.top + ( ( height - width ) / 2 );
        inner.bottom  = inner.top + width;
    }

    return( inner );
}

//-----
// Returns the largest rect that can be inscribed in the supplied bounding rect
// without changing the source rect's proportions, centered in the bounding rect's coordinates.

Rect do_Max_Inscribed_Rect( Rect src_rect, Rect bnd_rect )
{
    Rect dest_rect = {0,0,0,0};
    int src_width = src_rect.right - src_rect.left;
    int src_height = src_rect.bottom - src_rect.top;
    float src_w_to_h = (float)src_width / (float)src_height; // need to check div by zero
    int bnd_width = bnd_rect.right - bnd_rect.left;
    int bnd_height = bnd_rect.bottom - bnd_rect.top;
    float bnd_w_to_h = (float)bnd_width / (float)bnd_height; // need to check div by zero

    if( src_w_to_h == bnd_w_to_h )
    {
        dest_rect = bnd_rect;
    }
    else if( src_w_to_h > bnd_w_to_h )
    {
        // limit is width
        dest_rect.left = bnd_rect.left;
        dest_rect.right = bnd_rect.right;
        int dest_height = (float)bnd_width / src_w_to_h;
        dest_rect.top = bnd_rect.top + ( ( bnd_height - dest_height ) / 2 );
        dest_rect.bottom = dest_rect.top + dest_height;
    }
    else // if( src_w_to_h < bnd_w_to_h )
    {
        // limit is height
        dest_rect.top = bnd_rect.top;
        dest_rect.bottom = bnd_rect.bottom;
    }
}
```

```
int dest_width = (float)bnd_height * src_w_to_h;
dest_rect.left = bnd_rect.left + ( ( bnd_width - dest_width ) / 2 );
dest_rect.right = dest_rect.left + dest_width;
}

return( dest_rect );
}

//-----

Point do_Midpoint( Point p1, Point p2 )
{
    Point mp;

    mp.h = (p1.h + ((p1.h - p2.h) / 2));
    mp.v = (p1.v + ((p1.v - p2.v) / 2));

    return( mp );
}

//-----

#pragma mark -

//-----

void do_Draw_Styled_Text(short the_id, Rect *the_rect)
{
    Handle          theText;
    TEHandle        theTE;
    StScrpHandle    theStyle;

    theTE = TStyleNew(the_rect, the_rect);
    theText = GetResource('TEXT', the_id);
    theStyle = (StScrpHandle)GetResource('styl', the_id);
    HLock(theText);
    HidePen();
    if (theStyle != nil)
    {
        TStyleInsert(*theText, GetHandleSize(theText), theStyle, theTE);
        ReleaseResource((Handle)theStyle);
    }
    else
    {
        TEInsert(*theText, GetHandleSize(theText), theTE);
    }
    ShowPen();
    ReleaseResource(theText);

    TEUpdate(the_rect, theTE);
    TEDispose(theTE);
}

//-----

void do_Draw_Styled_Transparent_Text (short the_id, Rect *the_rect, short line)
{
    Handle          theText;
    TEHandle        theTE;
    StScrpHandle    theStyle;
    GWorldPtr       tWorld;
    CGrafPtr        saveCPort;
    GDHandle        saveGDevice;

    GetGWorld(&saveCPort, &saveGDevice);

    // tWorld = BuildOffScreenGWorld(8, the_rect, 0);
    if( noErr == do_Get_New_GWorld ( &tWorld, 32, the_rect, true ) )
    {
        LockPixels(GetGWorldPixMap(tWorld));
        SetGWorld(tWorld, nil);

        theTE = TStyleNew(the_rect, the_rect);
        theText = GetResource('TEXT', the_id);
        theStyle = (StScrpHandle)GetResource('styl', the_id);
        HLock(theText);
        HidePen();
        if (theStyle != nil)
        {
            TStyleInsert(*theText, GetHandleSize(theText), theStyle, theTE);
            ReleaseResource((Handle)theStyle);
        }
        else
        {
            TEInsert(*theText, GetHandleSize(theText), theTE);
        }
        TEPinScroll(0, line, theTE);
        ShowPen();
    }
}
```

```
        ReleaseResource(theText);

        TEUpdate(the_rect, theTE);
        TEDispose(theTE);

        SetGWorld(saveCPort, saveGDevice);
        CopyBits(    GetPortBitMapForCopyBits( (CGrafPtr)tWorld ),
                    GetPortBitMapForCopyBits( saveCPort ),
                    the_rect, the_rect, transparent, 0);
        UnlockPixels(GetGWorldPixMap(tWorld));
        DisposeGWorld(tWorld);
    }
}

//-----
#pragma mark -

//-----
//  · GetColorAndPenState
//
//  Gets the current drawing environment and stores the data in state. We copy pen and back
//  pix pats only if they are not type 0 (plain ol expanded black and white patterns).
//-----
//  With Carbon, we can use GetThemeDrawingState() and SetThemeDrawingState() instead.
/*
void GetColorAndPenState( ColorPenState* state )
{
    GrafPtr        curPort;

    GetPort( &curPort );

    state->pnPixPat = nil;
    state->bkPixPat = nil;

    state->colorPort = IsPortColor( curPort );

    state->bkPat = curPort->bkPat;
    state->bkColor = curPort->bkColor;
    state->fgColor = curPort->fgColor;

    if ( state->colorPort )
    {
        GetForeColor( &state->foreColor );
        GetBackColor( &state->backColor );

        // If the pen pattern is not an old style pattern,
        // copy the handle. If it is an old style pattern,
        // GetPenState below will save the right thing.

        if ( (*(CGrafPtr)curPort)->pnPixPat.patType != 0 )
        {
            state->pnPixPat = (CGrafPtr)curPort->pnPixPat;
        }

        // If the pen pattern is not an old style pattern,
        // copy the handle, else get the old pattern into
        // bkPat for restoring that way.

        if ( (*(CGrafPtr)curPort)->bkPixPat.patType != 0 )
        {
            state->bkPixPat = (CGrafPtr)curPort->bkPixPat;
        }
        else
        {
            state->bkPat = *(PatPtr)(*(CGrafPtr)curPort->bkPixPat.patData);
        }
    }

    GetPenState( &state->pen );
    state->textFont = curPort->txFont;
    state->textFace = curPort->txFace;
    state->textMode = curPort->txMode;
    state->textSize = curPort->txSize;
}
*/
//-----
//  · SetColorAndPenState
//
//  Sets the current drawing environment based on the data in state.
//-----
/*
void
SetColorAndPenState( ColorPenState* state )
{
    GrafPtr        curPort;

    GetPort( &curPort );
```

```
SetPenState( &state->pen );

if ( IsPortColor( curPort ) && state->colorPort )
{
    RGBForeColor( &state->foreColor );
    RGBBackColor( &state->backColor );

    if ( state->pnPixPat )
        PenPixPat( state->pnPixPat );

    if ( state->bkPixPat )
        BackPixPat( state->bkPixPat );
    else
        BackPat( &state->bkPat );
}
else
{
    BackPat( &state->bkPat );
    ForeColor( state->fgColor );
    BackColor( state->bkColor );
}

TextFont ( state->textFont );
TextFace ( state->textFace );
TextMode ( state->textMode );
TextSize ( state->textSize );
}
*/
//-----
//  · NormalizeColorAndPen
//
//  Sets up our environment to standard drawing fare.
//-----

void
NormalizeColorAndPen()
{
    NormalizeThemeDrawingState();
    /*
    RGBColor      black, white;
    Pattern        whitePat = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
    GrafPtr        curPort;

    GetPort( &curPort );

    black.red = black.green = black.blue = 0x0000;
    white.red = white.green = white.blue = 0xFFFF;

    if ( IsPortColor( curPort ) )
    {
        RGBForeColor( &black );
        RGBBackColor( &white );
    }
    PenNormal();
    BackPat( &whitePat );
    TextMode( srcOr );
    */
}
//-----

void
InverseNormalizeColorAndPen()
{
    RGBColor      black, white;
    Pattern        whitePat = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
    GrafPtr        curPort;

    GetPort( &curPort );

    black.red = black.green = black.blue = 0x0000;
    white.red = white.green = white.blue = 0xFFFF;

    if ( IsPortColor( curPort ) )
    {
        RGBForeColor( &white );
        RGBBackColor( &black );
    }
    PenNormal();
    BackPat( &whitePat );
    TextMode( srcOr );
}
//-----

void
GrayColorAndPen()
```

```
{
    RGBColor      black, white;
    Pattern       whitePat = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
    GrafPtr       curPort;

    GetPort( &curPort );

    black.red = black.green = black.blue = 0x3333;
    white.red = white.green = white.blue = 0xCCCC;

    if ( IsPortColor( curPort ) )
    {
        RGBForeColor( &black );
        RGBBackColor( &white );
    }
    PenNormal();
    BackPat( &whitePat );
    TextMode( srcOr );
}

//

#pragma mark -

//

void do_Track_Cursor_With_Square ( Point mouse_loc, short width, void (*callback)(void) )
{
    Point          old_mouse_loc, delta;
    Rect           the_rect;
    ThemeDrawingState state;
    //ColorPenState pen_state;
    RGBColor       black;
    Pattern        pat_black;

    black.red = black.green = black.blue = 0x0000;

    old_mouse_loc = mouse_loc;

    the_rect.left = mouse_loc.h - (width/2);
    the_rect.top = mouse_loc.v - (width/2);
    the_rect.right = the_rect.left + width;
    the_rect.bottom = the_rect.top + width;

    // GetColorAndPenState( &pen_state );
    GetThemeDrawingState( &state );

    PenMode ( patXor );
    PenSize ( 1, 1 );
    PenPat ( GetQDGlobalsBlack(&pat_black) );
    RGBForeColor ( &black );

    // FrameRect( &the_rect );
    PaintOval( &the_rect );

    #if TARGET_API_MAC_CARBON
        MouseTrackingResult tracking_result = kMouseTrackingMousePressed;
        while ( tracking_result != kMouseTrackingMouseReleased )
    #else
        while ( StillDown() )
    #endif
    {
        if ( EqualPt ( mouse_loc, old_mouse_loc ) == false )
        {
            PaintOval( &the_rect );
            delta.h = mouse_loc.h - old_mouse_loc.h;
            delta.v = mouse_loc.v - old_mouse_loc.v;

            OffsetRect( &the_rect, delta.h, delta.v );
            PaintOval( &the_rect );

            old_mouse_loc = mouse_loc;
        }

        if ( callback != NULL )
        {
            (*callback)();
        }
    }

    #if TARGET_API_MAC_CARBON
        TrackMouseLocation (NULL, &mouse_loc, &tracking_result);
    #else
        GetMouse(&mouse_loc);
        SystemTask ();
    #endif
}

// FrameRect( &the_rect );
```

```
PaintOval( &the_rect );

// SetColorAndPenState( &pen_state );
SetThemeDrawingState( state, true );
}

//
#pragma mark -

//
// Until Appearance 1.1 comes around, use the color codes
// defined in this file's accompanying header file.

void do_Set_Pen ( short which_color )
{
    GrafPtr      curPort;
    RGBColor      rgb_color;

    GetPort( &curPort );

    if ( IsPortColor( curPort ) )
    {
        switch ( which_color )
        {
            case PlatinumScrollbarActive:
            {
                if aqua
                if(1)
                {
                    GetThemeBrushAsColor( kThemeBrushButtonFrameActive, 32, true, &rgb_color );
                }
                else
                {
                    rgb_color.red =
                    rgb_color.green =
                    rgb_color.blue = 0;
                }
                break;
            }
            case PlatinumScrollbarInactive:
            {
                if aqua
                if(1)
                {
                    GetThemeBrushAsColor( kThemeBrushButtonFrameInactive, 32, true, &rgb_color );
                }
                else
                {
                    rgb_color.red =
                    rgb_color.green =
                    rgb_color.blue = 21845;
                }
                break;
            }
            default:
            {
                rgb_color.red =
                rgb_color.green =
                rgb_color.blue = 0;
                break;
            }
        }
    }

    RGBForeColor( &rgb_color );
    PenNormal();
}

//
// Use the Window Part Color Codes as defined
// in MacWindows.h and my_windows.h

void do_Set_Pen_To_WCTB_Color ( short which_color )
{
    GrafPtr      curPort;
    RGBColor      rgb_color, warning;

    warning.red = 0xFFFF;
    warning.green = warning.blue = 0x0000;

    GetPort( &curPort );

    if ( IsPortColor( curPort ) )
    {
        if ( do_Get_Window_RGBColor ( (WindowPtr)curPort, which_color, &rgb_color ) )
        {
            RGBForeColor( &rgb_color );
        }
    }
}
```

```
    }
    else
    {
        RGBForeColor( &warning );
    }
    PenNormal();
}
}

//
Boolean do_Get_Window_RGBColor ( WindowPtr window_ptr, short which_color, RGBColor *rgb_color )
{
    CTabHandle    color_table_handle = NULL;
    Boolean      got_color = false;

    color_table_handle = do_Get_Window_Color_Table ( window_ptr );
    if ( !color_table_handle )
        color_table_handle = do_Get_Window_Color_Table ( NULL );

    if ( color_table_handle )
    {
        got_color = do_Get_RGBColor_From_Color_Table ( color_table_handle, which_color, rgb_color );
    }

    return got_color;
}

//
// Given a Window pointer, this will return the color table associated with
// that window. Note, you can pass NULL to this routine to get the default
// window color table. Check out the documentation on GetAuxWin for details.
//
CTabHandle do_Get_Window_Color_Table ( WindowPtr window_ptr )
{
    CTabHandle    color_table_handle = NULL;
    //AuxWinHandle win_rec_handle;

    // (void) GetAuxWin ( window_ptr, &win_rec_handle );

    // if ( win_rec_handle != NULL )
    //     color_table_handle = (*win_rec_handle)->awCTable;

    color_table_handle = (CTabHandle)GetResource( 'wctb', 0 );

    return color_table_handle;
}

//
// Fills in a struct with the RGBColor asked for in the which_value param.
// Returns true if the color is filled in properly, and false if it isn't.
//
Boolean do_Get_RGBColor_From_Color_Table ( CTabHandle color_table, short which_color, RGBColor *rgb_color )
{
    Boolean      entry_found = false;
    short        index;

    if ( color_table != NULL )
    {
        for ( index = (*color_table)->ctSize; index > 0; index-- )
        {
            if ( (*color_table)->ctTable[index].value == which_color )
            {
                *rgb_color = (*color_table)->ctTable[index].rgb;
                entry_found = true;
                break;
            }
        }
    }

    return entry_found;
}

//
#pragma mark -

//
void do_Move_Mouse ( Point pt )
{
    gRawMouse->v = gTempMouse->v = pt.v;
    gRawMouse->h = gTempMouse->h = pt.h;
    LMSetCursorNew(true);
    // *gMouseNew = 1;
}
```

```
}  
  
//  
  
void do_Mouse_Couple ()  
{  
    *gCouple = 1;  
}  
  
//  
  
void do_Mouse_Decouple ()  
{  
    *gCouple = 0;  
}
```



```
// _____  
//                                     ©1998-2001 bergdesign inc.  
// _____
```

```
#ifndef __my_strings__  
#define __my_strings__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#include <string.h>  
#include <stdio.h>  
#include <ctype.h>  
  
#include "my_macros.h"  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
char *      do_p2c_str ( unsigned char * );  
unsigned char * do_c2p_str ( char * );  
  
unsigned char * do_p_strcpy ( unsigned char *dst, const unsigned char *src );  
char *          do_p2c_strcpy ( char *dst, const unsigned char *src );  
unsigned char * do_c2p_strcpy ( unsigned char *dst, const char *src );  
  
unsigned char * do_p_strcat ( unsigned char *dst, const unsigned char *src );  
char *          do_p2c_strcat ( char *dst, const unsigned char *src );  
unsigned char * do_c2p_strcat ( unsigned char *p_dst, const char *c_src );  
void            do_p_strerror ( unsigned char *dst, int err );  
void            do_p_strBCDcat ( unsigned char *dst, short bcd_ver_num );  
  
short          do_p_strcmp ( const unsigned char *a, const unsigned char *b );  
short          do_ci_p_strcmp ( const unsigned char *a, const unsigned char *b );  
short          do_p2c_strcmp ( const unsigned char *p_str, const char *c_str );  
  
char           *do_strdupr ( char *string );  
char           *do_strlwr ( char *string );  
  
#ifdef __cplusplus  
}  
#endif  
  
#endif /* __my_strings__ */
```



```
{
    if( (p_dst == NULL) || (p_src == NULL) )
        return( NULL );

    // First, we make length byte the same.
    // We could also write this as *dst = *src.
    p_dst[0] = p_src[0];

    // Copy *src bytes from src into dst and return the pointer
    return ( (unsigned char *)memcpy( (char *)(p_dst+1), (char *)(p_src+1), p_src[0] ) );
}
*/

//
// Copy a pstring to a cstring
char * do_p2c_strcpy ( char *c_dst, const unsigned char *p_src )
{
    register int    i = 0;

    if( (c_dst != NULL) && (p_src != NULL) )
    {
        // Copy chars from the pstring
        // until we reach the end of it
        while( i < p_src[0] )
        {
            c_dst[i] = p_src[i+1];
            i++;
        }

        // Terminate the c string
        c_dst[i] = '\0';
    }

    return( c_dst );
}

//
/*
char * do_p2c_strcpy ( char *c_dst, const unsigned char *p_src )
{
    unsigned char    length = p_src[0];

    if( (c_dst == NULL) || (p_src == NULL) )
        return( NULL );

    // First we need to terminate the C string at the
    // right byte for the pascal string being copied in.
    c_dst[length] = '\0';

    // Copy *src bytes from src into dst and return the pointer
    return ( (char *)memcpy( (c_dst), (char *)(p_src+1), p_src[0] ) );
}
*/

//
// Copy a cstring to a pstring
unsigned char * do_c2p_strcpy ( unsigned char *p_dst, const char *c_src )
{
    register int    i = 0;

    if( (p_dst != NULL) && (c_src != NULL) )
    {
        // Copy chars from the cstring until we reach
        // a null or the pstring limit of 255 chars
        while( (c_src[i] != '\0') && (i < 254) )
        {
            p_dst[i+1] = c_src[i];
            i++;
        }

        // Set the length of the p string
        p_dst[0] = i;
    }

    return( p_dst );
}

//
#pragma mark -

//
// Concatenate two pascal strings together.
unsigned char * do_p_strcat ( unsigned char *p_dst, const unsigned char *p_src )
{

```

```
short i, j;

if( (p_dst != NULL) && (p_src != NULL) )
{
    // Get the lengths of the strings.
    i = p_src[0];
    j = p_dst[0];

    // If the concatenated length is too long, figure
    // out how much of the appendage string we can use.
    if ( i + j > 255 )
        i = 255 - j;

    // Increment the length of the source string by the amount
    // of the appendage string that we can use.
    p_dst[0] += i;

    // Skip the pointers ahead to where the strings start.
    p_src++;
    p_dst += j + 1;

    // Count down from the max number of chars taken from the appended string,
    // putting the appendage string chars into the source string.
    while ( i-- > 0 )
    {
        *p_dst++ = *p_src++;
    }
}

return( p_dst );
}

//
// Concatenate a pascal string onto a c string
char * do_p2c_strcat ( char *c_dst, const unsigned char *p_src )
{
    unsigned long    c_len;

    if( (c_dst != NULL) && (p_src != NULL) )
    {
        // Get length of dst (NULL is not counted)
        c_len = strlen(c_dst);

        // Copy src[0] bytes from src to end of dst
        memcpy( (char *) (c_dst + c_len), (char *) (p_src+1), p_src[0] );

        // Calc length of new dst
        c_len = c_len + p_src[0];

        // Put in a terminating NULL character
        c_dst[c_len] = '\0';
    }

    return ( c_dst );
}

//
// Concatenate a c string onto a pascal string
unsigned char * do_c2p_strcat ( unsigned char *p_dst, const char *c_src )
{
    unsigned long    c_len, p_len;

    // unsigned char    p_src[256];
    // do_c2p_strcpy( p_src, c_src );
    // do_p_strcat( p_dst, p_src );

    if( (p_dst != NULL) && (c_src != NULL) )
    {
        // Get length of src (NULL not counted) and dst
        c_len = strlen(c_src);
        p_len = p_dst[0];

        // If the concatenated length is too long, figure
        // out how much of the appendage string we can use.
        if ( p_len + c_len > 255 )
            c_len = 255 - p_len;

        // Increment the length of the source string by the amount
        // of the appendage string that we can use.
        p_dst[0] += c_len;

        // Copy src[0] bytes from src to end of dst
        memcpy( (char *) (p_dst + 1 + p_len), c_src, c_len );
    }

    return( p_dst );
}
```

```
}

//
// This function concatenates an error number onto a pascal string.
// It's useful for putting the error info into a Mac dialog item.

void do_p_strerrcat ( unsigned char *dst, int err )
{
    char    err_txt[16];

    if( dst != NULL )
    {
        // Print the error number into a C string.
        sprintf ( err_txt, "%d", err );

        // Convert the C string into a pascal string.
        do_c2p_str ( err_txt );

        // Concatenate the new pascal string onto the existing pascal string.
        do_p_strcat ( dst, (unsigned char *)err_txt );
    }
}

void do_p_strBCDcat ( unsigned char *dst, short bcd )
{
    if( dst != NULL )
    {
        do_p_strerrcat( dst, ( ( HIBYTE( bcd ) >> 4 ) * 10 ) + ( HIBYTE( bcd ) & 0x0f ) );
        do_p_strcat( dst, "\p." );
        do_p_strerrcat( dst, LOBYTE( bcd ) >> 4 );
        do_p_strcat( dst, "\p." );
        do_p_strerrcat( dst, LOBYTE( bcd ) & 0x0f );
    }
}

//
#pragma mark -

//
// This is a pascal string compare function which is case sensitive.

short do_p_strcmp ( const unsigned char *a, const unsigned char *b )
{
    short    len = ( *a < *b ? *a : *b );
    short    lenEqual = ( *a > *b ) - ( *a < *b );
    short    equalityTest;

    while ( len )
    {
        ++a;
        ++b;

        equalityTest = ( *a > *b ) - ( *a < *b );

        if ( equalityTest )
            return ( equalityTest );

        --len;
    }

    return ( lenEqual );
}

//
// This is a pascal string compare function which is case insensitive.

short do_ci_p_strcmp ( const unsigned char *a, const unsigned char *b )
{
    int      i;
    register char    ac;
    register char    bc;

    if ( *a < *b )
    {
        return ( -1 );
    }
    else if ( *a > *b )
    {
        return ( 1 );
    }
    else
    {
        for ( i = 1; i <= *a; i++ )
        {
            ac = toupper ( a[i] );
            bc = toupper ( b[i] );
        }
    }
}
```

```
        if ( ac < bc )
            return ( -1 );

        if ( ac > bc )
            return( 1 );
    }

    return ( 0 );
}

//
// Compare a Pascal string with a C string
short do_p2c_strcmp ( const unsigned char *p_str, const char *c_str )
{
    char    tmp[256];

    do_p2c_strcpy( tmp, p_str );

    return( strcmp( tmp, c_str ) );
}

//
#pragma mark -

//
// Convert an entire c string to upper case
char *do_strupr(char *string)
{
    char *s;

    if ( string != NULL )
    {
        for (s = string; *s; ++s)
        {
            *s = toupper(*s);
        }
    }

    return string;
}

//
// Convert an entire c string to lower case
char *do_strlwr(char *string)
{
    char *s;

    if ( string != NULL )
    {
        for (s = string; *s; ++s)
        {
            *s = tolower(*s);
        }
    }

    return string;
}
```

```
//  
//  
//  
#ifndef __my_utilities_  
#define __my_utilities_  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC_  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <MacTypes.h>  
#include <Events.h>  
#include <ToolUtils.h>  
#include <Files.h>  
#include <Resources.h>  
#include <LowMem.h>  
#include <NumberFormatting.h>  
#include <Time.h>  
#endif  
#endif  
  
#include <string.h>  
#include <ctype.h>  
  
#include "my_macros.h"  
#include "my_dialogs.h"  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
#define SERIAL_NUMBER_KEY 3771529  
  
// This is the struct used by Photoshop for storage  
// of cross-platform information.  
struct OSInfo  
{  
    RestType      signature;  
    unsigned char  majorVersion;  
    unsigned char  minorVersion;  
    unsigned char  subVersion;  
    unsigned char  stage;  
    unsigned char  stageVersion;  
    unsigned char  os;  
};  
  
pascal Boolean      do_Is_Key_Down (unsigned short);  
pascal Boolean      do_Is_Bit_Set ( UInt32, int );  
  
pascal unsigned long do_To_Hash ( unsigned long, unsigned long );  
  
pascal unsigned long do_Encrypt_ULong (unsigned long, unsigned short *);  
pascal unsigned long do_Decrypt_ULong (unsigned long, unsigned short *);  
pascal void          do_ULong_2_String_W_Chkdigit (unsigned long , char * );  
pascal int           do_String_W_Chkdigit_2_ULong (unsigned long *, char * );  
  
pascal unsigned short do_Calc_1_CRC ( unsigned short );  
pascal unsigned short do_Calc_16_CRC ( unsigned char *, long, int );  
pascal unsigned long  do_Calc_32_CRC ( char *, long );  
  
pascal void          do_Get_App_Vers_Resource_As_CString ( char * );  
  
unsigned char        do_BCD_To_Hex ( unsigned char );  
unsigned char        do_Decimal_To_BCD( unsigned char );  
unsigned char        do_BCD_To_Decimal( unsigned char );  
  
unsigned long        do_Swap_Bytes_Of_Long( unsigned long );  
unsigned short       do_Swap_Bytes_Of_Short( unsigned short );  
  
pascal void          do_Print_Bytes_To_CString ( unsigned long *, char * );  
void                do_Print_Time_Date_To_CString ( char * );
```

```
        OSErr          do_Play_Sound ( short );
```

```
#ifdef __cplusplus  
}  
#endif
```

```
#endif /* __my_utilities__ */
```



```
//
//                                     ©1998-2001 bergdesign inc.
//
#include "my_utilities.h"

DECLARE_EXTERN_DEBUG_FILE_PTR;

//
//                                     get key code routine
//
pascal Boolean
do_Is_Key_Down ( unsigned short virtual_key_code )
// returns true if key defined is down
// uses the virtual key code, not the character key code
{
    KeyMap      theKeys;

    GetKeys ( theKeys );
    return ( BitTst( &theKeys, virtual_key_code ^ 0x07 ) );
}

//
//
pascal Boolean
do_Is_Bit_Set ( UInt32 x, int p )
{
    if ( ( x >> p ) & 0x01 )
        return true;
    else
        return false;
}

//
// simplified DES encryption hashing
//
pascal unsigned long
do_To_Hash ( unsigned long key, unsigned long input )
{
    unsigned long ia,ib,iswap,itmph=0,itmpl=0;

    ia  = (iswap=input) ^ 0xbaa96887L;
    itmpl = ia & 0xffff;
    itmph = ia >> 16;
    ib  = itmpl*itmpl+ ~(itmph*itmph);
    input = key ^ (((ia = (ib >> 16) | ((ib & 0xffff) << 16)) ^ 0x4b0f3b58L)+itmpl*itmph);
    key = iswap;

    ia  = (iswap=input) ^ 0x1e17d32cL;
    itmpl = ia & 0xffff;
    itmph = ia >> 16;
    ib  = itmpl*itmpl+ ~(itmph*itmph);
    input = key ^ (((ia = (ib >> 16) | ((ib & 0xffff) << 16)) ^ 0xe874f0c3L)+itmpl*itmph);
    key = iswap;

    ia  = (iswap=input) ^ 0x03bcde3cL;
    itmpl = ia & 0xffff;
    itmph = ia >> 16;
    ib  = itmpl*itmpl+ ~(itmph*itmph);
    input = key ^ (((ia = (ib >> 16) | ((ib & 0xffff) << 16)) ^ 0x6955c5a6L)+itmpl*itmph);
    key = iswap;

    ia  = input ^ 0x0f33d1b2L;
    itmpl = ia & 0xffff;
    itmph = ia >> 16;
    ib  = itmpl*itmpl+ ~(itmph*itmph);
    input = key ^ (((ib >> 16) | ((ib & 0xffff) << 16)) ^ 0x55a7ca46L)+itmpl*itmph);

    return input;
}

//
//
pascal unsigned long
do_Encrypt_ULong(unsigned long input,unsigned short *key)
{
    unsigned short low,high,ia,ib,iswap,itmph,itmpl;

    low = input & 0xffff;
    high = input >> 16;

    ia  = (iswap=low) ^ key[0];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + -(short)(itmph*itmph);
    low = high ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[1]) + (short)(itmpl*itmph));
    high = iswap;
}
```

```
    ia  = (iswap=low) ^ key[2];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + ~(short)(itmph*itmph);
    low = high ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[3]) + (short)(itmpl*itmph));
    high = iswap;

    ia  = (iswap=low) ^ key[4];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + ~(short)(itmph*itmph);
    low = high ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[5]) + (short)(itmpl*itmph));
    high = iswap;

    ia  = (iswap=low) ^ key[6];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + ~(short)(itmph*itmph);
    low = high ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[7]) + (short)(itmpl*itmph));
    high = iswap;

    return high << 16 | low;
}

//
```

```
pascal unsigned long
do_Decrypt_ULong(unsigned long input,unsigned short *key)
{
    unsigned short low,high,ia,ib,iswap,itmph,itmpl;

    low = input & 0xffff;
    high = input >> 16;

    ia  = (iswap=high) ^ key[6];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + ~(short)(itmph*itmph);
    high = low ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[7]) + (short)(itmpl*itmph));
    low = iswap;

    ia  = (iswap=high) ^ key[4];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + ~(short)(itmph*itmph);
    high = low ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[5]) + (short)(itmpl*itmph));
    low = iswap;

    ia  = (iswap=high) ^ key[2];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + ~(short)(itmph*itmph);
    high = low ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[3]) + (short)(itmpl*itmph));
    low = iswap;

    ia  = (iswap=high) ^ key[0];
    itmpl = ia & 0xff;
    itmph = ia >> 8;
    ib  = (short)(itmpl*itmpl) + ~(short)(itmph*itmph);
    high = low ^ (((ia = (ib >> 8) | ((ib & 0xff) << 8)) ^ key[1]) + (short)(itmpl*itmph));
    low = iswap;

    return high << 16 | low;
}

//
// these check digit routines, based on Verhoeff's dihedral group D5, finds:
// 100% of single errors
// 100% of adjacent transpositions
// 95% of twin errors (aa -> bb)
// 95% of jump transpositions (acb -> bca)
// 95% of jump twin errors (aca -> bcb)
```

```
pascal void
do_ULong_2_String_W_Chkdigit ( unsigned long number, char *ascii_digits )
{
    static int ip[10][8]={ {0,1,5,8,9,4,2,7},
                           {1,5,8,9,4,2,7,0},
                           {2,7,0,1,5,8,9,4},
                           {3,6,3,6,3,6,3,6},
                           {4,2,7,0,1,5,8,9},
                           {5,8,9,4,2,7,0,1},
                           {6,3,6,3,6,3,6,3},
                           {7,0,1,5,8,9,4,2},
                           {8,9,4,2,7,0,1,5},
```

```
static int ij[10][10]={ {9,4,2,7,0,1,5,8} },
                        {0,1,2,3,4,5,6,7,8,9},
                        {1,2,3,4,0,6,7,8,9,5},
                        {2,3,4,0,1,7,8,9,5,6},
                        {3,4,0,1,2,8,9,5,6,7},
                        {4,0,1,2,3,9,5,6,7,8},
                        {5,9,8,7,6,0,4,3,2,1},
                        {6,5,9,8,7,1,0,4,3,2},
                        {7,6,5,9,8,2,1,0,4,3},
                        {8,7,6,5,9,3,2,1,0,4},
                        {9,8,7,6,5,4,3,2,1,0} };

int j,k=0;
int str_length;

str_length = sprintf(ascii_digits,"%010u",(unsigned int)number);

// magic
for (j=0;j<str_length;j++)
{
    k=ij[k][ip[(ascii_digits[j]+2) % 10][7 & j]];
}

// more magic
for (j=0;j<=9;j++)
{
    if (!ij[k][ip[j][str_length & 7]])
        break;
}

// append the check digit
ascii_digits[str_length++] = j + 48;
ascii_digits[str_length] = 0;

// now add in some dashes

ascii_digits[13] = 0;
ascii_digits[12] = ascii_digits[10];
ascii_digits[11] = ascii_digits[9];
ascii_digits[10] = ascii_digits[8];
ascii_digits[9] = ascii_digits[7];
ascii_digits[8] = '-';
ascii_digits[7] = ascii_digits[6];
ascii_digits[6] = ascii_digits[5];
ascii_digits[5] = ascii_digits[4];
ascii_digits[4] = '-';
// ascii_digits[3] = ascii_digits[3];
// ascii_digits[2] = ascii_digits[2];
// ascii_digits[1] = ascii_digits[1];
// ascii_digits[0] = ascii_digits[0];
}

//
pascal int
do_String_W_Chkdigit_2_ULong ( unsigned long *number, char *ascii_digits )
{
static int ip[10][8]={ {0,1,5,8,9,4,2,7},
                        {1,5,8,9,4,2,7,0},
                        {2,7,0,1,5,8,9,4},
                        {3,6,3,6,3,6,3,6},
                        {4,2,7,0,1,5,8,9},
                        {5,8,9,4,2,7,0,1},
                        {6,3,6,3,6,3,6,3},
                        {7,0,1,5,8,9,4,2},
                        {8,9,4,2,7,0,1,5},
                        {9,4,2,7,0,1,5,8} };

static int ij[10][10]={ {0,1,2,3,4,5,6,7,8,9},
                        {1,2,3,4,0,6,7,8,9,5},
                        {2,3,4,0,1,7,8,9,5,6},
                        {3,4,0,1,2,8,9,5,6,7},
                        {4,0,1,2,3,9,5,6,7,8},
                        {5,9,8,7,6,0,4,3,2,1},
                        {6,5,9,8,7,1,0,4,3,2},
                        {7,6,5,9,8,2,1,0,4,3},
                        {8,7,6,5,9,3,2,1,0,4},
                        {9,8,7,6,5,4,3,2,1,0} };

int j,k=0;
int str_length;
char temp_str[32];

*number = 0;

// copy without the dashes

temp_str[0] = ascii_digits[0];
temp_str[1] = ascii_digits[1];
temp_str[2] = ascii_digits[2];
temp_str[3] = ascii_digits[3];
```

```
temp_str[4] = ascii_digits[5];
temp_str[5] = ascii_digits[6];
temp_str[6] = ascii_digits[7];
temp_str[7] = ascii_digits[9];
temp_str[8] = ascii_digits[10];
temp_str[9] = ascii_digits[11];
temp_str[10] = ascii_digits[12];
temp_str[11] = 0;

str_length = strlen(temp_str);

// magic
for (j=0;j<str_length;j++)
{
    if(!isdigit(temp_str[j]))
        return 0;

    k=i+j[k][ip[(temp_str[j]+2) % 10][7 & j]];
}

// see if it checked out ok
if(k==0)
{
    // take off the check digit
    temp_str[str_length-1] = 0;

    sscanf(temp_str,"%u",(unsigned int *)number);

    return 1;
}
else
    return 0;
}

//
pascal unsigned short
do_Calc_1_CRC ( unsigned short crc )
{
    int i;
    unsigned short ans;

    ans = (crc ^ 0 << 8);

    for (i=0;i<8;i++)
    {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }

    return ( ans );
}

//
pascal unsigned short
do_Calc_16_CRC ( unsigned char *bufptr, long len, int jrev )
{
    static unsigned short icrctb[256],init=0;
    static unsigned char rchr[256];
    unsigned short cword;
    unsigned long j;
    static unsigned char it[16]={0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15};

    if(!init)
    {
        init=1;
        for (j=0;j<256;j++)
        {
            icrctb[j] = do_Calc_1_CRC(j << 8);
            rchr[j] = (unsigned char)(it[j & 0xF] << 4 | it[j >> 4]);
        }
    }

    cword = 0;

    for (j=0;j<len;j++)
        cword = icrctb[(jrev < 0 ? rchr[bufptr[j]] : bufptr[j]) ^ HIBYTE(cword)] ^ LOBYTE(cword) << 8;

    return ( jrev >= 0 ? cword : rchr[HIBYTE(cword)] | rchr[LOBYTE(cword)] << 8 );
}

//
pascal unsigned long
```

```
do_Calc_32_CRC ( char *data, long length )
{
    unsigned short first_crc, second_crc;

    // this function is a bit of a hack, using two 16 bit crc's to get one 32 bit one
    // it has no error detection capabilities, but it should make a nearly unique crc for every file

    first_crc = do_Calc_16_CRC((unsigned char *)data,(unsigned long)length,1);
    second_crc = do_Calc_16_CRC((unsigned char *)data,(unsigned long)length,-1);

    return ( (unsigned long)first_crc << 16 | (unsigned long)second_crc );
}

// _____

pascal void
do_Get_App_Vers_Resource_As_CString ( char *string )
{
    struct OSInfo    version;
    Handle           vers_handle = NULL;
    Ptr              vers_pointer = NULL;
    short            current_resource_ID;
    unsigned char    minor_and_sub;
    char             stage[8];
    char             temp[8];

    // Get the current resource file so we can restore it later.
    current_resource_ID = CurResFile();

    // Set the resource file to the application.
    UseResFile( LMGetCurApRefNum() );

    // Get a handle to the 1st vers resource of the application.
    vers_handle = Get1Resource ( 'vers', 1 );

    // If there was no problem...
    if ( !ResError() )
    {
        HLock( vers_handle );
        vers_pointer = *vers_handle;
        if ( vers_pointer != NULL )
        {
            version.majorVersion = (unsigned char)*vers_pointer++;
            version.majorVersion = do_BCD_To_Hex ( version.majorVersion );
            DEBUG_VAR_PRINT("Major Version is %0X",version.majorVersion);

            // The minor version and sub version share a byte.
            minor_and_sub = (unsigned char)*vers_pointer++;

            // Since the minor and sub versions share a byte, we have to separate them.
            version.minorVersion = minor_and_sub >> 4; // top 4 bits = vers2
            version.minorVersion = do_BCD_To_Hex ( version.minorVersion );
            DEBUG_VAR_PRINT("Minor Version is %0X",version.minorVersion);

            version.subVersion = minor_and_sub & 0x0F; // bottom 4 bits = vers3
            version.subVersion = do_BCD_To_Hex ( version.subVersion );
            DEBUG_VAR_PRINT("Sub Version is %0X",version.subVersion);

            version.stage = (unsigned char)*vers_pointer++;
            DEBUG_VAR_PRINT("Development stage is %0X",version.stage);

            version.stageVersion = (unsigned char)*vers_pointer;
            DEBUG_VAR_PRINT("Stage version is %0X",version.stageVersion);
        }
        HUnlock ( vers_handle );
        ReleaseResource ( vers_handle );
        vers_handle = NULL;
        vers_pointer = NULL;
    }

    UseResFile ( current_resource_ID );

    if ( version.stage == 0x20 )
    {
        sprintf ( stage, "d" );
    }
    else if ( version.stage == 0x40 )
    {
        sprintf ( stage, "a" );
    }
    else if ( version.stage == 0x60 )
    {
        sprintf ( stage, "b" );
    }
    else if ( ( version.stage == 0x80 ) && ( version.stageVersion != 0 ) )
    {
        sprintf ( stage, "fc" );
    }
}
```

```
else
{
    stage[0] = '\0';
}

string[0] = '\0';
sprintf ( string, "%u.%u", version.majorVersion, version.minorVersion );

temp[0] = '\0';
if ( version.subVersion != 0 )
    sprintf ( temp, ".%u", version.subVersion );

strcat ( string, temp );
strcat ( string, stage );

temp[0] = '\0';
if ( strlen ( stage ) != 0 )
    sprintf ( temp, "%u", version.stageVersion );

strcat ( string, temp );
}

//
// In Binary Coded Decimal (BCD) format, every four bits are
// used to represent one decimal digit.

unsigned char do_BCD_To_Hex ( unsigned char bcd_char )
{
    unsigned char    hex_char;

    hex_char = ( ( bcd_char / 16 ) * 10 ) + ( bcd_char - ( ( bcd_char / 16 ) * 16 ) );

    return ( hex_char );
}

//
// To convert from a 2's Complement number to a BCD number,
// take the value of the digit in the ten's place, store it
// in the leftmost four bits of a byte, then add to it the
// value of the digit in the one's place.

unsigned char do_Decimal_To_BCD( unsigned char n )
{
    return ((n / 10) << 4) + (n % 10);
}

//
// Converting from BCD to decimal requires multiplying the
// value in the leftmost four bits by 10 and adding the
// value of rightmost four bits to the result.

unsigned char do_BCD_To_Decimal( unsigned char n )
{
    return ((n >> 4) * 10) + (n & 0x0f);
}

//
// Routine to byte swap a long, just returns on big-endian

unsigned long do_Swap_Bytes_Of_Long( unsigned long value )
{
#ifdef _LITTLE_ENDIAN
    char *ptr, c;

    ptr = (char *)&value;
    c = *(ptr+1);
    *(ptr+1) = *(ptr+2);
    *(ptr+2) = c;
    c = *ptr;
    *ptr = *(ptr+3);
    *(ptr+3) = c;
#endif _LITTLE_ENDIAN

    return(value);
}

//
// Routine to byte swap a short, just returns on big-endian

unsigned short do_Swap_Bytes_Of_Short( unsigned short value )
{
#ifdef _LITTLE_ENDIAN
    char *ptr, c;

    ptr = (char *)&value;
```

```
c = *(ptr+1);
*(ptr+1) = *(ptr);
*ptr = c;

#endif _LITTLE_ENDIAN

return(value);
}

// _____

pascal void
do_Print_Bytes_To_CString ( unsigned long *bytes, char *string )
{
float          result;
unsigned long   kilobyte = 1024;
unsigned long   megabyte = 1024 * 1024;
unsigned long   gigabyte = 1024 * 1024 * 1024;

    if ( *bytes >= gigabyte )
    {
        result = (float)(*bytes) / (float)gigabyte;
        sprintf ( string, "%.1f GB", result );
    }
    else if ( *bytes >= megabyte )
    {
        result = (float)(*bytes) / (float)megabyte;
        sprintf ( string, "%.1f MB", result );
    }
    else if ( *bytes >= kilobyte )
    {
        result = (float)(*bytes) / (float)kilobyte;
        sprintf ( string, "%.1f KB", result );
    }
    else
    {
        result = (float)(*bytes);
        sprintf ( string, "%.1f Bytes", result );
    }
}

// _____

void do_Print_Time_Date_To_CString ( char *time_string )
{
time_t          now_t;
struct tm       *now_tm;
size_t          max_chars = 255;

    now_t = time ( NULL );
    now_tm = localtime ( &now_t );

    strftime ( time_string, max_chars, "%b %d, %Y %I:%M:%S %p", now_tm );
}

// _____
//                                     play a sound

OSErr do_Play_Sound ( short snd_res_id )
{
Handle          snd_handle;
OSErr           err;

    snd_handle = GetResource ( 'snd ', snd_res_id );
    err = SndPlay ( NULL, (SndListHandle)snd_handle, true );
    ReleaseResource ( snd_handle );

    return ( err );
}
```

```
//  
// _____ ©1998 bergdesign inc.  
// _____  
  
#ifndef __my_windows__  
#define __my_windows__  
  
#ifndef ACCESSOR_CALLS_ARE_FUNCTIONS  
#define ACCESSOR_CALLS_ARE_FUNCTIONS 1  
#endif  
#ifndef OPAQUE_TOOLBOX_STRUCTS  
#define OPAQUE_TOOLBOX_STRUCTS 1  
#endif  
  
#ifdef __APPLE_CC__  
#include <Carbon/Carbon.h>  
#else  
#if TARGET_API_MAC_CARBON  
#include <Carbon.h>  
#else  
#include <Appearance.h>  
#include <Menus.h>  
#include <MacWindows.h>  
#include <Events.h>  
#include <Gestalt.h>  
#include <Memory.h>  
#include <Processes.h>  
#include <Resources.h>  
#include <LowMem.h>  
#include <ToolUtils.h>  
#include <Errors.h>  
#include <Quickdraw.h>  
#include <Dialogs.h>  
#endif  
#endif  
  
#include "my_macros.h"  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
// _____  
  
// Dialog routines  
  
RgnHandle      GetDialogContentRegion ( DialogRef );  
RgnHandle      GetDialogStructureRegion ( DialogRef );  
void           GetDialogPortRect ( DialogRef, Rect * );  
  
// Window routines  
Boolean        WindowIsDialog ( WindowRef );  
Boolean        WindowIsModal ( WindowRef );  
Boolean        WindowIsFloater ( WindowRef );  
  
void           DeactivateFloatersAndFirstDocumentWindow ( void );  
void           ActivateFloatersAndFirstDocumentWindow ( void );  
void           do_Activate_Window ( WindowRef, Boolean );  
  
// Process stuff  
static Boolean  IsFrontProcess ( void );  
  
// Misc stuff  
Rect           do_Get_Display_Bounds_From_GDHandle ( GDHandle );  
  
#ifdef __cplusplus  
}  
#endif  
  
#endif /* __my_windows__ */
```



```
//  
//  
//  
//  
//  
// Change History (most recent first):  
//  
// 6/98 Brock Brandenburg  
//   · Change window procs to Appearance Manager compliant versions.  
//   · Change include Types.h to MacTypes.h  
//   · Change include GestaltEqu.h to Gestalt.h  
// 11/00 Brock Brandenburg  
//   · Remove almost everything  
//  
// Based on code from Apple's WindowExtensions.c from Dean Yu and Dave Johnson  
//  
  
# include "my_windows.h"  
  
DECLARE_EXTERN_DEBUG_FILE_PTR;  
  
//  
  
RgnHandle GetDialogContentRegion(DialogRef theDialog)  
{  
    RgnHandle region_handle = NULL;  
  
    GetWindowRegion( GetDialogWindow(theDialog), kWindowContentRgn, region_handle );  
    return ( region_handle );  
}  
  
//  
  
RgnHandle GetDialogStructureRegion(DialogRef theDialog)  
{  
    RgnHandle region_handle = NULL;  
  
    GetWindowRegion( GetDialogWindow(theDialog), kWindowStructureRgn, region_handle );  
    return ( region_handle );  
}  
  
//  
  
void GetDialogPortRect(DialogRef theDialog, Rect *portRect)  
{  
    CGrafPtr the_port = GetWindowPort( GetDialogWindow(theDialog) );  
    GetPortBounds( the_port, portRect );  
}  
  
//  
  
#pragma mark -  
  
//  
// WindowIsDialog  
//  
// Determines if a window is a dialog based upon the value  
// of its windowKind.  
//  
//  
  
Boolean WindowIsDialog ( WindowRef window_ref )  
{  
    WindowClass    window_class = 0L;  
  
    GetWindowClass ( window_ref, &window_class );  
  
    if (    window_class == kAlertWindowClass ||  
          window_class == kMovableAlertWindowClass ||  
          window_class == kModalWindowClass ||  
          window_class == kMovableModalWindowClass )  
    {  
        DEBUG_VAR_PRINT("Called WindowIsDialog(%#010X) -> true",window_ref);  
        return true;  
    }  
    else  
    {  
        DEBUG_VAR_PRINT("Called WindowIsDialog(%#010X) -> false",window_ref);  
        return false;  
    }  
}  
  
//  
// WindowIsModal  
//  
// Determines if a window is modal based upon the value of its windowKind  
// and window variant.
```

```
//  
  
Boolean WindowIsModal ( WindowRef window_ref )  
{  
    UInt32 window_features = 0L;  
  
    GetWindowFeatures ( window_ref, &window_features );  
  
    if ( window_features & kWindowIsModal )  
    {  
        DEBUG_VAR_PRINT("Called WindowIsModal(%#010X) -> true",window_ref);  
        return true;  
    }  
    else  
    {  
        DEBUG_VAR_PRINT("Called WindowIsModal(%#010X) -> false",window_ref);  
        return false;  
    }  
}  
  
//  
// WindowIsFloater  
//  
// Determines if a window is a floater based upon the value of its windowKind.  
//  
  
Boolean WindowIsFloater ( WindowRef window_ref )  
{  
    WindowClass window_class = 0L;  
  
    GetWindowClass ( window_ref, &window_class );  
  
    if ( window_class == kFloatingWindowClass )  
    {  
        DEBUG_VAR_PRINT("Called WindowIsFloater(%#010X) -> true",window_ref);  
        return true;  
    }  
    else  
    {  
        DEBUG_VAR_PRINT("Called WindowIsFloater (%#010X) -> false",window_ref);  
        return false;  
    }  
}  
  
//  
  
#pragma mark -  
  
//  
// DeactivateFloatersAndFirstDocumentWindow  
//  
// Send deactivate events to all visible floating windows and the active document  
// window. This routine is called before a modal dialog is presented.  
//  
  
void DeactivateFloatersAndFirstDocumentWindow ( void )  
{  
    WindowRef window_ref = FrontNonFloatingWindow();  
  
    DEBUG_PRINT("Called DeactivateFloatersAndFirstDocumentWindow()");  
  
    if ( NULL != window_ref )  
    {  
        ControlHandle root_control = NULL;  
  
        OSStatus err = GetRootControl( window_ref, &root_control );  
        if ( noErr == err && NULL != root_control )  
        {  
            DeactivateControl ( root_control );  
        }  
    }  
  
    HideFloatingWindows();  
}  
  
//  
// ActivateFloatersAndFirstDocumentWindow  
//  
// ActivateFloatersAndFirstDocumentWindow should be called after a modal dialog  
// is dismissed. If the application is in the background when this routine is  
// called (like when a moveable modal progress dialog was up and then disappears)  
// this routine calls SuspendFloatingWindows to hide any visible floating windows  
// instead.  
//  
  
void ActivateFloatersAndFirstDocumentWindow ( void )  
{  
    DEBUG_PRINT("Called ActivateFloatersAndFirstDocumentWindow()");  
}
```

```
// Sanity check: if an app "nests" modal dialogs, and calls
// ActivateFloatersAndFirstDocumentWindow every time a dialog is dismissed
// then this could be called inappropriately. Assume that if a modal window
// is up, caller didn't really mean it :-)
if( WindowIsModal( FrontWindow() ) )
    return;

// See if the this process is in the background. If it is, then the floating
// windows should be hidden instead of reactivated, so SuspendFloatingWindows()
// is called instead.
if( IsFrontProcess() )
{
    ShowFloatingWindows();
}
else
{
    HideFloatingWindows();
}
}

//
void do_Activate_Window ( WindowRef the_window, Boolean activate )
{
    OSStatus      err = noErr;
    ControlHandle  root_control = NULL;

    DEBUG_VAR_PRINT("Activating window %#010X",the_window);
    DEBUG_EXTRA_VAR_PRINT(" = %d",activate);

    if ( activate )
    {
        SelectWindow( the_window );
        SetPortWindowPort( the_window );
    }

    err = GetRootControl( the_window, &root_control );
    if ( noErr == err && NULL != root_control )
    {
        if ( activate )
            ActivateControl ( root_control );
        else
            DeactivateControl ( root_control );
    }
}

//
#pragma mark -

//
Boolean IsFrontProcess ( void )
{
    ProcessSerialNumber  currentPSN;
    ProcessSerialNumber  frontPSN;
    OSStatus             getFrontProcessResult;
    OSStatus             getCurrentProcessResult;
    Boolean              isSameProcess = false;

    // Compare this process and the front process
    getFrontProcessResult = GetFrontProcess(&frontPSN);
    getCurrentProcessResult = GetCurrentProcess(&currentPSN);

    if ((getFrontProcessResult == noErr) && (getCurrentProcessResult == noErr))
        SameProcess(&frontPSN, &currentPSN, &isSameProcess);

    return isSameProcess;
}

//
#pragma mark -

//
Rect do_Get_Display_Bounds_From_GDHandle ( GDHandle gdh )
{
    SInt8      gdh_state = 0;
    Rect       bounds_rect = { 0, 0, 0, 0 };
    OSStatus   err = noErr;

    // The right way to get a display's bounds.
    // Do not use the bounds of the gdPMap of the gdh.

    gdh_state = HGetState ( (Handle)gdh );
    err = MemError();
}
```

```
if ( err == noErr )
{
    HLock ( (Handle)gdh );
    err = MemError();
    if ( err == noErr )
    {
        bounds_rect = (*(gdh))->gdRect;
        HSetState ( (Handle)gdh, gdh_state );
    }
}

return ( bounds_rect );
}
```